Pojam i definicija komunikacije

Pojam komunikacionog procesa

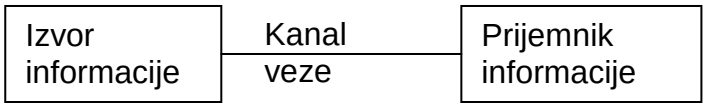
Prijenos podataka, vijesti ili obavijesti između ljudi i mehanizama ili pak jednih i drugih međusobno, promatran u globalnom - najširem smislu kao komunikacije, predstavlja vrlo složen proces.U vezi s tim problem je multidisciplinaran i u njegovom proučavanju dodirujemo ili pak direktno "zadiremo" u različite naučne discipline kao: psihologija, biologija, filozofija, telekomunikacije i drugo.

Globalno: **komunikacije obuhvataju načine, mehanizme i medije uključene u prijenos informacija.** Komunikacije moraju odgovoriti na tri pitanja :

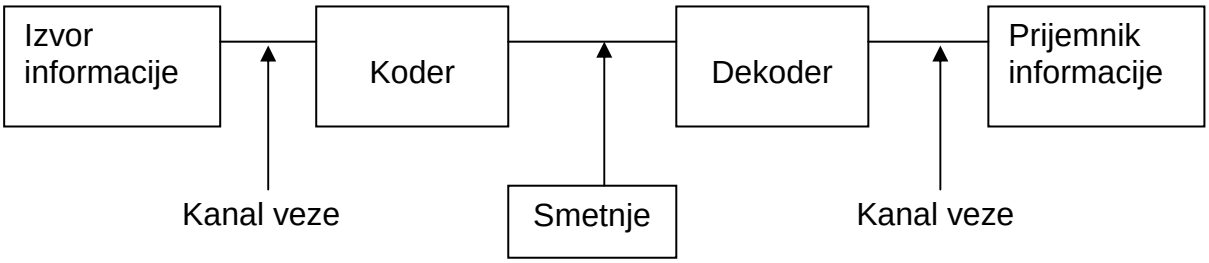
- Kojom tačnošću mogu biti preneseni simboli od kojih je sačinjena poruka (***sintaksa***).
- Koliko precizno preneseni simboli nose željeno značenje ili smisao poruke (***semantika***).
- S kakvim uspjehom prenesena poruka kroz svoje značenje usmjerava na neku željenu aktivnost onoga kome je bila namijenjena (***pragmatika***).

Svaki komunikacijski sistem može se općenito prikazati modelom prikazanim na slici:
Model komunikacijskog sistema :

- izvor informacije
- prijemnik informacije
- kanal veze



Prema Shannonu (Šanonu), osnovne komponente modela procesa komunikacije su :



Shannonov model procesa komunikacije

Kada se na stranu predaje kao i prijemnu stranu komunikacijskog sistema postavlja – povezuje računar tada govorimo o **računarskoj mreži**

Dva su osnovna načina komunikacije u računarskoj mreži:

1. komunikacija posredstvom uređaja MODEM (Modulator- Demodulator)
2. mrežna kartica – veza prema lokalnoj mreži (Local AreaNetwork)

Informacija predstavlja preslikavanje stanja jednog subjekta u stanje drugog subjekta.

Npr. Ako jedan čovjek govori drugim ljudima engleskim jezikom, onda on svoje određeno znanje ' preslikava ' na njih. Jedan od slušalaca ga razumije, a to je onaj slušalac koji zna engleski jezik, dok drugi ne razumiju. Na taj način prenos informacija kod slušalaca je izvršen na različite načine.

Iz primjera je vidljivo da je pojam informacije vezan za proces prenošenja informacija između subjekata koji učestvuju u komunikaciji.
U procesu komunikacije postoje dva subjekta (izvor i prijemnik informacija). Subjekti su povezani kanalom veze .
Subjekti u procesu komuniciranja mogu biti : čovjek, mašina, knjiga, ...
Veza između subjekata koji komuniciraju može biti izražena : govorom, pismom, slikom, muzikom, ...
Pri procesu komunikacije između subjekata, mogu se pojaviti i smetnje (buka ili šum)

Shannonov model komunikacije

Komunikaciona veza može biti :

1. Kooperativna ili **dvosmjerna** komunikaciona veza.
2. Nekooperativna ili **jednosmjerna** komunikaciona veza.

Kooperativna komunikacija je takav način komunikacije gdje izvor i prijemnik naizmjenično mjenjaju informacije.
Npr. Razgovor dva čovijeka .

Nekooperativna komunikacija je takav način komunikacije u kojoj nema zamjene uloga odnosno gdje nema povratnih informacija između subjekata komunikacije. Npr. Praćenje TV programa

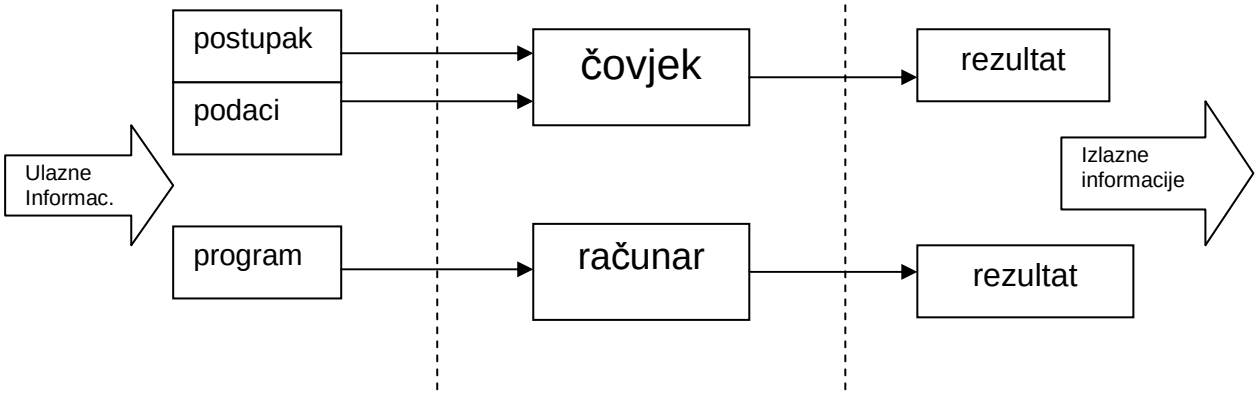
U teoriji informacija , definirana su tri nivoa posmatranja komunikacionog procesa, mada među njima nema oštre granice. To su :

- ***Tehnički nivo*** – bavi se određenim aspektima prenosa informacija komunikacionim kanalima kao i tačnošču prenesene poruke . Ovaj nivo obezbjedjuje da se poruka u neizmjenjenom obliku prenese do izvora pa do prijemnika . Kad se jedna poruka formulira, pitanje je kako će se ona prenijeti kanalom veze i da li će biti smetnji.

- ***Semantički nivo*** – razmatra da li je prenesena poruka ima ili nema određeno značenje za prijemnik, odnosno ima li poruka smisla i da li je prijemnik primljenu poruku u potpunosti razumo.

-
- ***Efektivni nivo*** - razmatra primanje informacije i njihov značaj pri donošenju odluka. Ovaj nivo ispituje kako neka informacija može da doprinese da prijemnik donenese određenu odluku koju bez te informacije nije mogao donijeti.

Komunikacija čovjek – računar



Za komunikaciju čovjeka i računara služe programski (umjetni) jezici.

Umjetni jezik je jezik koji je izmišljen posebno za formuliranje programa i predstavlja komunikaciono sredstvo između čovjeka i računara.
Svaki se viši programski jezik sastoji od određenih simbola i znakova, naziva i brojeva. Kao što postoje određena pravila govornih jezika , tako se i kod viših jezika za programiranje govori o ***sintaksi*** (o pravilnom nizanju simbola, znakova i naziva) i ***semantici*** (o interpretaciji upotrebljenih simbola).

Računar razumije samo jedan jezik , i to **mašinski jezik** tako da programi napisani u bilo kojem programskom jeziku se prevode na jezik računara, tj mašinski jezik.
Obično se početni program napisan u nekom programskom jeziku naziva ***izvorni kod*** , a njegov prevedeni oblik u mašinski jezik naziva ***objektni kod***.

Prevođenje programa iz izvornog koda u objektni vrši specijalni program koji se zove ***kompiler***.. Za svaki programski jezik postoji poseban kompajler.

Primjeri algoritama u prirodnom jeziku (kojeg slušamo svakodnevno) pokazali su da se predstavljanje malo složenijih zadataka na ovaj način nije pogodno.Iz tog razloga dolazi do pojave umjetnih (izvedenih) jezika. U svakom slučaju da bi se napisao neki jezik ,mora se koristiti neki drugi jezik kojeg poznajemo.

Metode napredne komunikacije čovjek - računar

Postoje različite metode i sredstva kojim se obezbjeđuje napredna komunikacija između čovjeka i računara.
Metode konverzacije sa računarom se mogu podijeliti u 6 glavnih tipova:

- metode komandi
- menija
- upita-odgovora
- popunjavanja formulara
- funkcionalnih ključeva
- sličica

Najsavremeniji metod napredne komunikacije je metod korištenje vještačke inteligencije.O vještačkoj inteligenciji će biti više riječi nešto kasnije.

Računarski programi
(Programska podrška)

Računarske programe (SOFTWARE) čine svi PROGRAMI s kojima je digitalni računarski sistem snabdjeven i koji se svakodnevno nadopunjuju i razvijaju. Izrađuju ih proizvođači računarskog sklopovlja ili za to specijalizirane firme (poput Microsoft-a), dobivaju se uz računarsku opremu ili se mogu pronaći na tržištu ili ih, prema svojim potrebama, korisnik sam izrađuje. Programska podrška uobičajeno se isporučuje na disketama ili na CD-u uz popratnu dokumentaciju kako je instalirati u sistem i upute o korištenju.

U osnovi programska podrška dijeli se na:

PODRŠKU SISTEMA (sistemski softver) –
koju uglavnom isporučuje prodavač Digitalnih računarskih sistema, a omogućava iskorištavanje svih resursa računarskog sistema (hardware-a).

Sistemski programi mogu biti:

Nadzorni program **: je sistemski programi koji je smješten u glavnoj memoriji računara. Često se nadzorni program naziva *i operativni sistem*.**

Neovisni sistemski program: **moguće je izvoditi po potrebi . To su kompilatori, programi prevodioci, i pomoćni programi za upravljanje podacima. Sistemski programi su pisani isključivo za određeni sistem.**

KORISNIČKU PODRŠKU (aplikativni softver)-
koju čine svi programi koje je korisnik računara kupio radi rješavanja zadaća,kao programi za obradu teksta, baze podataka i drugo, ili ih je sam izradio za to predviđenim programskim alatima.

Program je skup naredbi računaru napisanih po utvrđenim pravilima po kojima računar vrši zadanu obradu podataka.

PROGRAM u suštini predstavlja REDOSLJEDNI niz instrukcija, koje računar izvršava da bi obavio zadaću prema zadanom postupku rješavanja tj. po izrađenom ALGORITMU.

Programski jezik je skup naredbi i pravila sintakse kojima se kreiraju programske instrukcije (programi).

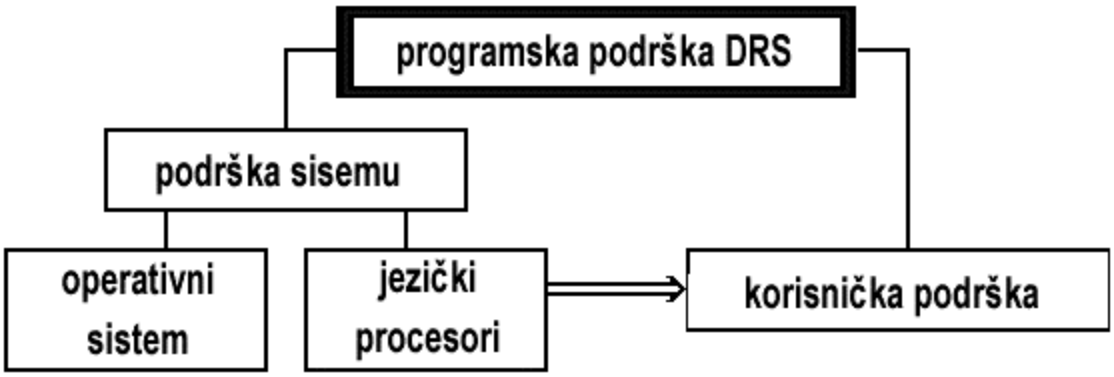
Računar postupa prema instrukcijama. Instrukcije predstavljaju sastavni dio programa kojeg je napisao programer. Programske instrukcije naređuju računaru šta treba da odradi.

Programska podrška sistema u suštini upravlja radom sklopova i dijeli se u dvije grupe:

OPERATIVNI SISTEM - koji obuhvaća sve programe koji kontroliraju izvršavanje programa korisnika, njihov redosljed i smještaj u memoriji i vrše razne operacija sa njima (promjena imena ili brisanje na primjer). Kontrolira "kretanje" bit-ova kroz računar i pruža informacije o aktivnostima računara i ostalih uređaja u njemu i oko njega. NE SLUŽI za kreiranje korisničke programske podrške. Različit je za različite vrste računara. Poznati su MS-DOS,WINDOWS,OS/2,UNIX,...

JEZIČKI PROCESORI - su programi koji prevode čovjeku razumljive instrukcije i komande u binarne zapise razumljive računaru. Predstavljaju prevoditelja između čovjeka i računara u postupku KREIRANJA aplikacijskog software-a. Dakle, to su alati sa kojima će se napraviti program za crtanje slika ili program za računovodstvo i knjiženje ili nešto drugo.Nazivaju se i programski jezici.To su PASCAL,C,BASIC,FORTRAN,COBOL i drugi.

Na slijedećoj slici prikazana je podijela sistema programske podrške i veze između pojedinih elemenata.



GENERACIJE PROGRAMSKIH JEZIKA

Ne postoji jednoznačno određenje podjele programskih jezika na generacije, ali se oni najčešće dijele kako slijedi:

Jezici prve generacije (engl. *first generation languages*): mašinski jezici
Sintaksa ovih jezika bazirana je na tzv mašinskim instrukcijama.
Mane: velika mogućnost greške u postupku programiranja

Jezici druge generacije (engl. *second generation languages*): asembleri

Jezici treće generacije (engl. *third generation languages*): programski jezici opće namjene (npr.FORTRAN,ALGOL,COBOL,BASIC, Pascal, C, C++)
Ovi jezici se nazivaju *problemski orjentirani proceduralni jezici.*
Pojavljuje se potreba kompilacije (prevođenja) jezika.

Jezici četvrte generacije (engl. *fourth generation languages* - 4GL jezici): programski jezici uske namjene (npr. dBASE, SQL, PostScript)
korisnik utvrđuje šta traži od računarskog sistema , a sistem sam iznalazi način zadovoljenja korisničkih zahtjeva. To su problemski orjentirani jezici (neproceduralni jezici).

Mašinski jezici i asembleri nazivaju se **niži programski jezici** i ovisni su o računaru.

Programski jezici četvrte generacije su viši programski jezici poznati su pod nazivom 4GL (fourth generation languages), a karakterizira ih način komunikacije sličan prirodnim jezicima. Na primjer, tipična naredba 4GL jezika glasi:

FIND ALL RECORDS WHERE NAME IS "SMITH"

4GL jezici se uglavnom koriste za rad s bazama podataka.(Informix, C/BASE 4GL, Oracle)

GENERACIJE PROGRAMSKIH JEZIKA

GENERACIJA	TIP JEZIKA	PRIMJER NAREDBI
Prva	**Machine(mašinski)**	**10010001**
druga	**Assembly(asembleri)**	**ADD 210(8, 13),02B(4, 7)**
treća	**Proceduralni**	**Overtime: 5 0**
četvrta	**Problemski**	**FIND NAME 5 "JONES"**
peta	**Natural(prirodni)**	**IF patient is dizzy, THEN check temperature and blood pressure** (Ako pacijent ima vrtoglavicu, onda izmjeri temperaturu i krvni tlak)

Niži programski jezici

Mašinski jezik

Mala brzina izvršenja programa, pisanih na višem programskom jeziku, i veličina memorije koju taj program zauzima glavni su razlozi za programiranje na ***mašinskom jeziku.***

Ta ograničenja dolaze do izražaja u velikim programima za upravljanje , regulaciju, mjerenje, crtanje, i obradu velikog broja podataka.

Prednost viših programskih jezika je lakoća i preglednost u njegovom pisanju.
Mašinski program je sačinjen od linije mašinskih naredbi i svaka od njih je smještena na odgovarajući način u posebnu memorijsku lokaciju. Adresa te memorijske lokacije označava položaj naredbe u programu. Svaka linija mašinskog programa se sastoji od naredbe i ,eventualno, podatka.Naredba je sastavljena od koda operacije i operanda.

Zbog velikog broja kodova naredbi uveden je simbolički mašinski jezik (asembler) u kome su naredbe u svojim mnemoničkim oznakama .

Mašinski jezici su osnovni jezici koje računar razumije, posebno prilagođeni svakom tipu procesora, pisani pomoću binarnih cifara (1,0).
To je jedini oblik programa koji razumije računar, te se svaki drugi oblik programa mora prevesti u mašinski jezik prije nego se izvodi.

Asembler

Asembleri su slični mašinskim jezicima, ali jednostavniji za korištenje, jer su cifre zamijenjene mnemoničkim nazivima (simbolima) koji se lako pamte (napr. ADD za sabiranje ili SUB za oduzimanje)

Primjer naredbi napisanih u asemblerskom jeziku:

```
MOV A,#03
CAL TIME_DELAY
CLR P1.6
CLR P1.6
```

Programer, koji piše program u programskom jeziku asembleru mora dobro poznavati građu računara. Program za prevođenje asembler prevodi jednu po jednu naredbu programskog jezika asembler u odgovarajuću binarnu naredbu mašinskog jezika.

Primjer :

	adrese		mnemonika	KODOVI	
	dec	heksadec		Decim.	Heksadec.
A=100	30000	7530	LD A,100	62,100	3E,64
HL=23609	30002	7532	LD HL,23609	33,57,92	21 39 5C
Poke HL,A	30005	7535	LD(HL),A	119	77

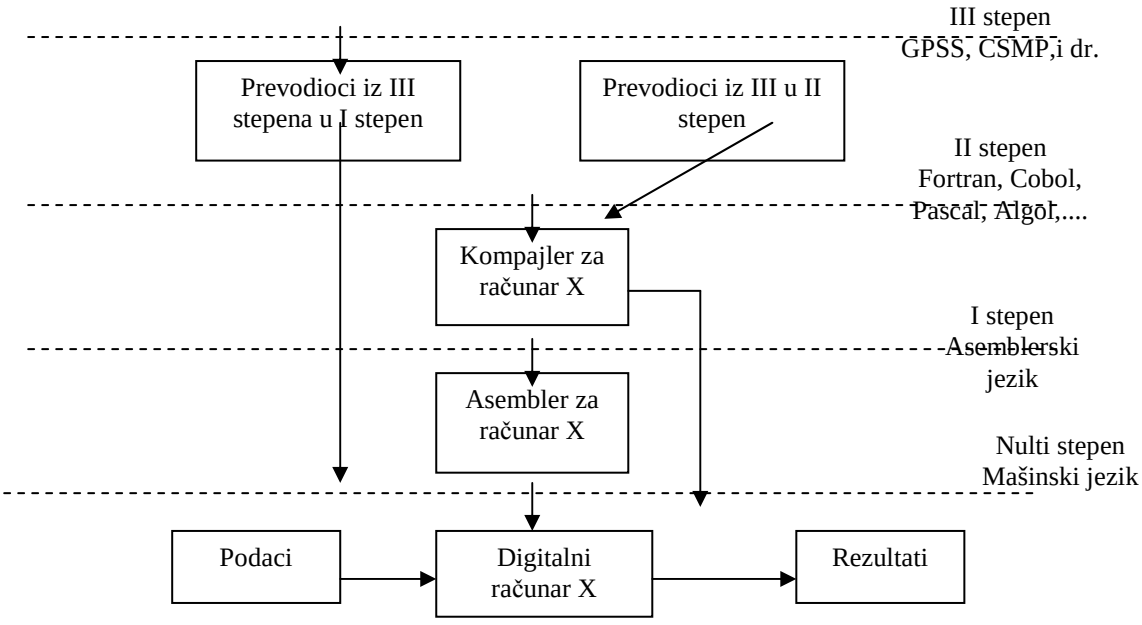
POJAŠNJENJE PRIMJERA :

1. Na adresama 30000 se nalazi mašinska naredba koja u registar A stavlja sadržaj 100. Naredba se nalazi u 2 memorijske lokacije , u prvoj je smješten kod operacije (62) , a u drugoj je operand 100.
2. Kod treće operacije je 119 u memorijskoj lokaciji 30005. Pomoću nje se sadržaj registra A prebacuje u memorijsku lokaciju čija je adresa u paru registra HL.
   (POKE n,m)(u memorijskoj lokaciji sa adresom n, upisuje se broj m)

Viši programski jezici

Digitalni računar «razumije» samo jednu vrstu naredbi, naredbe u obliku binarnih brojeva prikazanih na registrima. Kod *asemblerskih jezika* se prevođenje u binarne brojeve obavlja *prevodiocima* tako da se jedna simbolička naredba prevede u jednu naredbu u binarnom obliku (mašinski programski jezik)

Viši programski jezici imaju takve naredbe da se jedna naredba u višem jeziku prevede u veći broj naredbi u binarnom obliku. To prevođenje obavlja ***kompajler*** .
Kompajler ovisi o programskom jeziku.
Prevodioci mogu biti jednostavni ali i složeni , pa prema tome zauzimaju mali ili vrlo veliki prostor u memoriji.
Zbog toga manji računari rade najčešće samo u asemblerskom jeziku.
Kako se odnose pojedine vrste programskih jezika prikazano je na slici:



Asemblerski jezici čine I stepen računarskih jezika. Moguće je da za jedan asemblerski jezik i isti računar postoji više raznih asemblera.
Za programiranje u mašinskom jeziku, osim poznavanja problema koji se želi riješiti, potrebno je dobro poznavati hardver koji će se upotrebiti,odnosno njegove instrukcije te je potrebno imati određeno iskustvo u programiranju tom mašinom.

Drugi stepen čine jezici koji zahtjevaju posjedovanje kompajlera.
Drugi stepen čine jezici opće namjene:
FORTRAN- FORmula TRANslation
ALGOL – ALGOritamski jezik
COBOL – COmmon Business Oriented Language
PL/1 - Programming Language One- pokušaj da se napravi univerzalni jezik za rješavanje problema koji rješavaju Fortran, Cobol , Algol,..

Treći stepen čine jezici specifične namjene ili problemsko orijentirani jezici.
CSMP (COmmon System Modeling Program) - simulacija kontinuiranih procesa
GPSS (General Purpose Simulation System) – simulacija diskretnih procesa
Za prevođenje tih jezika u mašinski potreban je složeniji prevodilac. Prevođenje se može obaviti u tri ili dva koraka.

Viši programski jezici mogu raditi na tzv. 'konverzacijski način' .
Pri tom načinu rada, računaru se izdaje naredba a on je odmah *prevede i izvede*, te se rad obavlja u vidu *dijaloga*.
Taj se način rada posebno razvio uz upotrebu tzv. ' tajm šering (time sharing) ' sistema, kod kojeg jedan računar poslužuje mnogobrojne korisnike raspoređujući svoje vrijeme između njih. Pri tome se svakom korisniku čini da računar radi samo za njega.
Prevodilački programi koji omogućuju rad na konverzacijski način nazivaju se «*interpreteri*»

Viši programski jezici (engl. high level languages, HLL) su problemski orijentirani jezici koji dozvoljavaju pisanje programa načinom bliskim korisniku. Zbog toga su jednostavniji za pisanje i održavanje, a istovremeno ih je moguće prenijeti na različite računarske platforme. Svaki program pisan u višem programskom jeziku potrebno je prije izvođenja prevesti u mašinski jezik. Pojedina naredba pisana u višem programskom jeziku prilikom prevođenja raščlanjuje se na više mašinskih naredbi. Programi prevodioci koje u tu svrhu koristimo mogu biti kompajleri (compiler) i interpreteri. Prvi viši programski jezici razvili su se 50-tih godina.

Svi programski jezici, osim mašinskog jezika, nazivaju se *SIMBOLIČKIM JEZICIMA*.

PREDNOSTI viših programskih jezika u odnosu na mašinski jezik:
- jednostavnost pisanja programa,
- njihova razumljivost,
- prenosivost.

NEDOSTACI viših programskih jezika u odnosu na mašinski jezik:
- relativno sporije izvođenje,
- nemogućnost direktnog nadzora sklopovskih dijelova računara.

Prednosti viših programskih jezika u odnosu na mašinski jezik tako su velike da se danas mašinski jezik kod personalnih računara koristi samo iznimno i kada je to prijeko potrebno.

Značajna prednost viših programskih jezika je ta što se lakše nauče i upotrebljavaju nego asemblerski. Pri pravljenju programa upotrebom viših programskih jezika se ne mora voditi računa o organizaciji hardverskih komponenata, kao što treba činiti pri upotrebi asemblerskog jezika. Jedna naredba viših programskih jezika obavlja više poslova, te lakše shvata već napisan program. To je veoma značajno pri traženju i otklanjanju grešaka u programu (debugging). Programi viših programskih jezika su dosta kraći nego u asemblerskom jeziku.
Jedna od najbitnijih prednosti viših programskih jezika je mogućnost da se jedan program upotrebi na više različitih računara. Vrijeme rješavanja zadataka je znatno kraće ,upotrebom viših programskih jezika nego kad se upotrebe asemblerski jezici.

***Nedostaci*:**

Jedan od nedostataka viših programskih jezika je vrijeme potrebno da se neki program kompajlira. To kompajliranje ne daje djelotvorni program u mašinskom jeziku. (program zauzima mnogo prostora u memoriji i vremena za izvođenje)
Neki se problemi ne mogu riješiti upotrebom viših programskih jezika.
Može se reći da za masovnu obradu podataka se upotrebljavaju isključivo viši programski jezici , a za postavljanje i pravljenje hardverskih i softverskih sistema , asembleri su nezamjenjivi.

Primjer:

PROGRAM ZA NZD

Dati su programi za NZD u Pascalu i u asembleru:

```
VAR
  a, b: INTEGER;

BEGIN

  WriteLn('Prvi broj');
  ReadLn(a);

  WriteLn('Drugi broj');
  ReadLn(b);

  WHILE (a <> b) DO

    IF (a > b) THEN
      a := a - b
    ELSE
      b := b - a;

  WriteLn('NZD je:', a);

END.
```

```
DATA  SEGMENT WORD PUBLIC

      ASSUME DS:DATA

DATA  ENDS

CODE  SEGMENT BYTE PUBLIC
      ASSUME CS:CODE
      MOV AX, 32
      MOV BX, 128

START:
      CMP AX, BX
      JE KRAJ
      JG AX_JE_VECI
      SUB BX, AX
      JMP START

AX_JE_VECI:
      SUB AX, BX
      JMP START

KRAJ:
      JMP KRAJ

CODE  ENDS
      END
```

Kod viših jezika za programiranje treba definisati veći broj pravila za izraze koji su dopušteni (odnosno sintaksu).

OZNAČAVANJE VARIJABLI

Varijable imaju imena koja se sastoje od jednog ili više slova i brojeva , ali tako da prvi znak bude slovo.
Upotrebljavaju se slova engleske abecede (26 slova).
Razlikuju se cjelobrojne i realne varijable, znakovne varijable, logičke varijable,...

OZNAČAVANJE OPERACIJA

+ - sabiranje ; - - oduzimanje ; x – množenje ; / - dijeljenje

Umjesto znaka **x** većinom se upotrebljava znak ***** .
Razlomljeni brojevi se obično označavaju sa decimalnom tačkom a ne decimalnim zarezom (3.215)

STANDARDNE FUNKCIJE

Kompajler sadrži podprograme za funkcije (drugi korijen, sinus, kosinus, tangens, arkus tangena, za stepene i logaritamske funkcije, za određivanje cijelog dijela nekog broja i sl.)

PRIDJELJIVANJE VRIJEDNOSTI VARIJABLI

Varijabla kojoj se pridjeljuje vrijednost izražena je eksplicitno sa lijeve strane znaka = ili :=
Npr. A=b+c ili A:= b+c

SKOKOVI

Najvažnije su tri vrste skokova : bezuslovni skok, uslovni skok, skok na potprogram

PETLJE

Predstavljaju ponavljanje pojedinih manjih dijelova programa više puta uzastopno.
(DOK , DO , BROJAČKA STRUKTURA)

Jezici prevoditelji

Centralna jedinica za obradu podataka može prihvatati program samo u mašinskom jeziku. Program u višem jeziku potrebno prije izvođenja prevesti na mašinski jezik (engl. *machine code, object code*) pomoću INTERPRETERA ili KOMPAJLERA.

INTERPRETER je program koji izvršava druge programe, tj. INTERPRETER je jezički prevoditelj koji svaku naredbu izvornog programa (engl. *source code*) u trenutku izvođenja programa prevodi u binarni oblik mašinskog jezika (engl. *machine code, object code*).

KOMPAJLER (kompilator, engl. *compiler*) je program za prevođenje izvornog programa (engl. *source code*) u mašinski jezik (engl. *machine code, object code*) samo jednom i to tokom prevođenja ili kompajliranja.
Svojstva interpretera i kompajlera:

INTERPRETER:

Prednosti: prevodi naredbu po naredbu omogućava otkrivanje sintaksnih grešaka i interaktivno ispravljanje, tj. otkrivanje grešaka pri svakom izvođenju programa i automatsko prevođenje pri popravkama programa.
Nedostaci: sporiji rad od kompajlera, nužnost davanja izvornog programa korisniku.
Primjeri programa koje prevodi interprete: Lotus, dBase, GW Basic.

KOMPAJLER:

Prednosti: brži rad od interpretera, zaštićen izvorni program.
Nedostaci: odvojenost prevedenog i izvornog programa (izvršni EXE fajl).

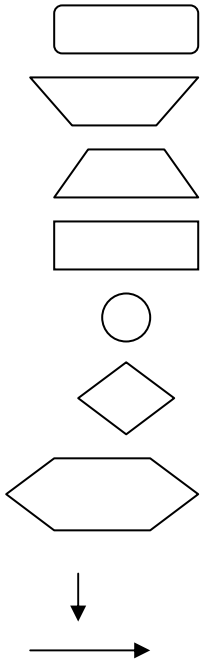
Primjeri programa koje prevodi kompajler: C, Turbo Pascal, Clipper.

ALGORITMI

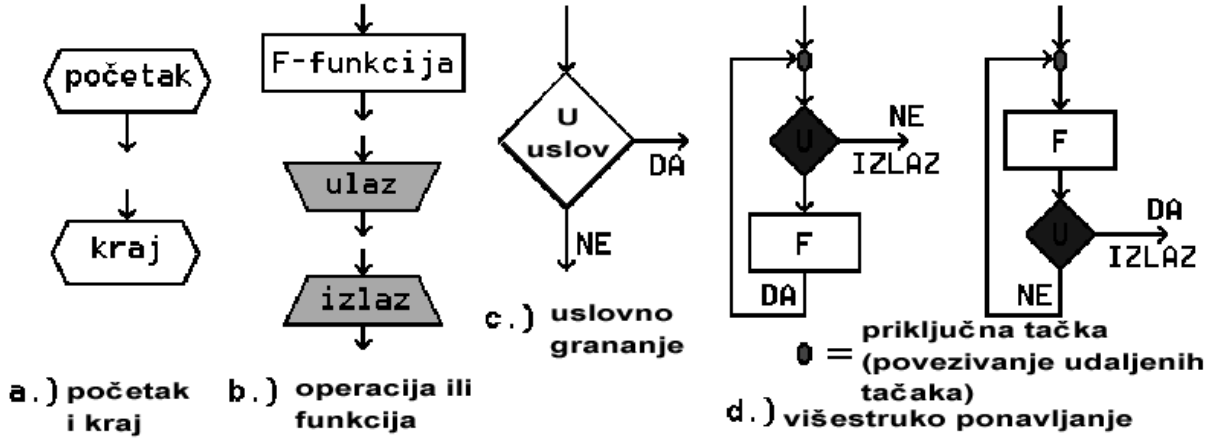
Cjelokupni rad računara zasniva se u osnovi na slijedećem: procesor prima podatke od ulaznog uređaja, obradi ih i izvrši i rezultat pošalje na izlazni uređaj kako bi ih prikazao korisniku. Svaki procesor raspolaže sa skupom instrukcija koje su kodirane BROJEM. Izvršavanje programa je u suštini kopiranje brojeva u radnu memoriju i njihovo očitavanje i izvršavanje od strane procesora. Niz brojeva međusobno poslaganih u jednu svrsishodnu cjelinu čini PROGRAM, a postupak izrade te cjeline pomoću jednog od jezičkih procesora naziva se PROGRAMIRANJE. Niz instrukcija jezičkog procesora, napisan u jednom od editor-a, koje će se potom procesoru prevesti da ih može izvršiti, mora biti logičan i usklađen s MODELOM odvijanja zadataka koji se naziva ALGORITAM. Npr. postupak kuhanja jela nije ništa drugo do poštivanje algoritma nazvanog recept.

Dakle, algoritam predstavlja logički niz radnji koje treba izvršiti da se od početnih postavki dođe do željenog rezultata. Algoritam se može posmatrati kao alat za rješavanje zadanih problema. On opisuje postupak za povezivanje ulaznih i izlaznih podataka. Algoritam mora biti sasvim precizno definiran postupak, bez dvosmislenosti i nedorečenosti. a izrađuje se povezivanjem logičkih struktura koje se prikazuju grafičkim oblicima (dijagramom toka), najčešće prema ANSI standardu:

Grafički simboli dijagrama toka



- početak ili kraj
- ulaz (učitavanje) podataka
- izlaz (ispis) rezultata
- blok obrade podataka (izvršenje naredbe)
- povezivanje udaljenih dijelova dijagrama ili oznaka koraka
- logička odluka na osnovu vrijednosti nekog izraza
- modifikacije programa
- redoslijed izvođenja operacija (programa)



Prva logička struktura je najprostija, označava početak i kraj algoritma (programa).

Druga logička struktura koristi se za operacije koje imaju jedan ulaz i jedan izlaz kao funkcije, na primjer za Y=f(X)=2*X ulaz je neka veličina X a funkcija na izlazu daje izračunatu vrijednost za taj X. Za ulaz X=2 izlaz će biti Y=4. Takve strukture nanizane jedna izad druge tvore niz (SEKVENCU) i predstavljaju osnovnu (linijsku) strukturu algoritma.

Uslovno grananje (SELEKCIJA) je izbor između dva moguća puta odvijanja programa, na primjer za prethodnu funkciju f(x)=2*X i uslov f(X)>3 program će se za X=1 nadalje odvijati prema izlazu DA a za X=2 prema izlazu NE (struktura *IF-THEN-ELSE*; ako-onda-odnosno). Moguće je izmjeniti izlaze na slici što u suštini ne mijenja odluku već samo smjer na slici.

Višestruko ponavljanje (ITERACIJA) može biti proračun funkcije nakon zadovoljenog uslova (struktura *DO-WHILE*; ispitaj pa radi) ili odluka nakon proračuna funkcije (struktura *DO-UNTIL*; radi pa ispitaj). Ako je na primjer ulaz X=0, funkcija f(x)=X+1 i uslov f(X)>3, lijeva struktura po ispitivanju prosljeđuje X=0 na izlaz, a desna struktura će povećavati X za jedan sve dok X ne poprimi vrijednost X=4 koja se prosljeđuje na izlaz, dakle broji do 4 . Ako je ulaz X=7 desna struktura povećati će ga na 8 i proslijediti na izlaz, a lijeva će neprestano povećavati vrijednost X za jedan dok ne dođe do prepunjenja registra mikroprocesora (OVERFLOW). Dakle, u zadavanju uslova treba biti obazriv tako da uslovi budu usklađeni s vrijednošću na ulazu i s tipom petlje ponavljanja. Lijeva petlja ispravno će raditi za f(x)<3 i proslijediti će X=3 na izlaz.

Algoritam zadatka složen od logičkih struktura prikazanih grafičkim oblicima naziva se ***DIJAGRAM TOKA*** ili ***BLOK DIJAGRAM*** i predstavlja grafičko rješenje programske zadaće. Na osnovu dijagrama toka piše se niz instrukcija jezičkog procesora.

Za izradu algoritma (razradu logike) programa poslužićemo se tekstualnim izražavanjem. Bez strogo definiranog pseudo jezika (kakav se često koristi u udžbenicima pri opisu algoritama), koristićemo se rečenicama govornog jezika. Te rečenice biti će "strukturirane" po određenom pravilu (koje će biti jasno iz primjera) i biti će napisane tako, da će se moći pretvoriti u jednu ili više naredbi C ili nekog drugog programskog jezika. Naučiti razraditi algoritam nekog problema, ili slobodnije rečeno, naučiti "misliti" na način kako to od nas računar "očekuje", osnovni je preduslov za uspješno programiranje u bilo kojem programskom jeziku. Savjetujemo, zato, posebno početnicama u programiranju, da ovom poglavlju posvete temeljitu pažnju.

Primjer

Razraditi logiku programa koji će najprije učitati broj n. Ako broj n nije prirodan broj ispisati poruku o grešci i ponoviti učitavanje. Učitavati redom, jedan za drugim, n brojeva x i naći i ispisati najveći od njih xmax.

Rješenje:

Postavi n=0
Sve dok n nije prirodan broj čini
    Učitaj broj n
    Ako n nije prirodan broj ispiši poruku
    o pogrešno zadanom broju n
    Kraj Sve dok
    Učitaj prvi broj xmax
    Za svaki i=2 do n čini slijedeće
        Učitaj broj x
        Ako je x > xmax postavi xmax=x
        Kraj Za svaki
Ispiši xmax

Komentar rješenja:

Prikazano rješenje je jedno od više mogućih. Odabrano je upravo ovo, da bude pregledno i razumljivo. Rečenice su pisane s glagolima u imperativu ("Učitaj", "Postavi"). To naglašava smisao rečenice kojom korisnik (programer) naređuje automatu tj. računaru da izvrši zadanu radnju. "Učitaj broj n" je naredba računaru da prihvati (s nekog ulaznog uređaja, tastature, diska) zadani podatak. Naredbe će se odvijati u redoslijedu kako su napisane ako samom naredbom nije drugačije naređeno. Očito je, da će se nakon naredbe "Učitaj broj n" obaviti naredba "Ako n...". No nakon reda "Kraj Sve dok" ponavlja se petlja "Sve dok.." ako je zadani uslov "n nije prirodan broj" ispunjen ili se obavlja iduća naredba ako je taj uslov nije ispunjen, odnosno n je prirodan broj. Također se petlja (iteracija) "Za svaki i=2 do n čini slijedeće" ponavlja n-1 puta da bi se nakon tih n-1 ponavljanja dobio najveći broj xmax koji se nakon obavljene iteracije ispisuje naredbom "Ispiši xmax".

Algoritam ne daje sasvim detaljno naznaku kako će se riješiti svaka naredba. Tako druga naredba ispituje da li je n prirodan broj. Način kako se to rješava, ovisi o programskom jeziku. U C-u, gdje n možemo definirati kao cijeli broj, biće dovoljno ispitati, da li je on veći od nule. Rješenje ovog primjera u C-u dato je u primjeru 2.1.

Zadatak Razraditi logiku programa koji će najprije učitati broj n. Ako n nije prirodan broj, ispisati o tome poruku i ponoviti učitavanje. Učitati n brojeva i naći i ispisati njihovu aritmetičku sredinu.

Zadatak Razraditi logiku programa koji će učitavati jedan za drugim niz brojeva. Kraj niza brojeva označen je brojem -99999. Naći i ispisati aritmetičku sredinu učitanih brojeva (broj -99999 ne uzeti u obzir).

Zadatak Razraditi logiku programa koji će učitavati jedan za drugim niz brojeva. Kraj niza označen je brojem 9. Naći i ispisati aritmetičku sredinu učitanih brojeva uzimajući u obzir samo one brojeve koji su veći ili jednaki 2 i manji ili jednaki pet.

Zadatak Razraditi logiku programa koji će učitati broj n. Taj broj mora biti prirodan i manji od 34. Ako taj uslov nije zadovoljen ponoviti učitavanje broja n. Izračunati i ispisati n! (faktorijel od n=1*2*3*..*(n-1)*n).

Zadatak Razraditi logiku programa koji će učitati brojeve m i n. Oba broja moraju bit prirodna i m mora biti veći od n. Ako taj uvjet nije ispunjen ponoviti učitavanje. Naći i ispisati binomni koeficijent b jednak m povrh n tj.

$$b = \frac{m!}{n!\,(m-n)!}$$

Zadatak Razraditi logiku programa koji će učitati dva broja n1 i n2. Oba broja trebaju biti prirodna. Ako taj uslov nije ispunjen, ponoviti učitavanje. Ako je n2 < n1, zamijeniti n1 sa n2. Naći i ispisati sumu drugih korjena svih neparnih brojeva od n1 do n2.

Primjer

Učitati prirodan broj n. Naći da li je učitani broj prost ili ne i o tome ispisati odgovarajući tekst.

Rješenje:

Učitaj broj n
Ako je n < 4 čini
    Ispiši poruku da je n prost broj"
    Zaustavi dalji račun
    Kraj Ako je

Ako je n paran čini
    Ispiši poruku da n nije prost broj jer je djeljiv s 2
    Zaustavi dalji račun
    Kraj Ako je

Postavi m = drugi korjen iz n
Za svaki i=3 do m u koraku po 2 učini slijedeće:
    Ako je n djeljivo s i čini
        Ispiši poruku da n nije prost jer je djeljiv s i
        Zaustavi dalji račun
        Kraj Ako je
    Kraj Za svaki i
Ispiši poruku da je n je prost broj

Komentar rješenja:

Broj je prost ako nije djeljiv ni sa jednim brojem osim s 1 i samim sobom. Učitani broj n (za koji pretpostavljamo da je prirodan), ako je veći od 1 i manji od 4 onda je prost broj (2 ili 3). Ako je n paran (i veći od 3) tada nije prost i djeljiv je s 2.

Neparan broj ispitujemo da li je prost tako, da ga dijelimo sa svakim neparnim brojem i to od 3 do m pri čemu je m drugi korjen iz n (broj koji nije djeljiv ni sa jednim brojem manjim od m sigurno nije djeljiv ni sa brojevima većim od m). Naredbom za iteraciju "Za svaki i=3 do m u koraku po 2" i poprima vrijednosti neparnih brojeva jer je se i povećava za 2 u svakoj narednoj iteraciji.

Zadatak Razraditi logiku programa koji će najprije učitati broj n. Broj n mora biti prirodan i veći od 3. Ako taj uslov nije ispunjen, ispisati poruku o grešci i ponoviti učitavanje. Naći i ispisati namanji prosti broj veći od n.

Zadatak Razraditi logiku programa koji će učitati broj n koji mora biti veći od 10. Ako taj uslov nije ispunjen, ispisati poruku o grešci i prekinuti rad programa. Naći i ispisati najveći prosti broj manji od n.

Zadatak Razraditi logiku programa koji će učitati brojeve n1 i n2. Ako je n1 veće od n2, zamijeniti njihove vrijednosti. Naći i ispisati sve proste brojeve u intervalu n1 do n2.

Zadatak Razraditi logiku programa koji će najprije učitati prirodan broj n i realan broj x. Program treba naći i ispisati sumu prvih n članova niza s=1+x/1!+x**2/2!+..+x**n/n! (** označava stepenovanje).

Zadatak Razraditi logiku programa koji će učitati realne brojeve a, b i c i naći i ispisati rješenja x1 i x2 kvadratne jednačine a*x**2 + b*x + c = 0. Ako jednačina nema realnih rješenja (b*b-4*a*c<0), ispisati odgovarajuću poruku i ne računati rješenja.

Zadatak Razraditi logiku programa koji će učitati brojeve a, b i c. Sva tri broja moraju biti manja od 20 i zadovoljavati uslov a + b > c. Ako ti uslovi nisu zadovoljeni, ispisati odgovarajuću poruku i ponovo učitati brojeve a, b i c. Ako su brojevi ispravno zadani, naći površinu p trokuta sa stranicama a, b i c. Koristiti Heronovu formulu p=sqrt(s*(s-a)*(s-b)*(s-c)). (Sa sqrt je označen drugi korjen izraza u zagradi, a s=(a+b+c)/2).

Zadatak Razraditi logiku programa koji će učitati broj n. Taj broj mora biti veći od nule i manji od 10. Ako taj uslov nije ispunjen, ispisati odgovarajuću poruku i ponoviti učitavanje. Naći i ispisati srednju vrijednost svih parnih brojeva od n do 2000 ali ne uzimajući u obzir one parne brojeve, koji su djeljivi sa 13.

Primjer

Razraditi logiku programa koji će najprije učitati broj n, koji mora biti prirodan broj manji od 1000. Ako n ne zadovaljava postavljeni uslov, ispisati poruku i ponoviti učitavanje. Ako n zadovoljava postavljeni uslov, učitati n brojeva, poredati ih (sortirati) po rastućem redoslijedu i ispisati tako poredane brojeve.

Rješenje:

Definiraj vektor (jednodimenzionalnu matricu) v od 999 brojeva
Postavi n=0
Sve dok n nije prirodan broj manji od 1000 čini
    Učitaj broj n
    Ako je n > 999 ili n<=0 ispiši poruku za ponovno zadavanje broja n
    Kraj Sve dok
Za svaki i= 0 do n-1 učitaj v[i]. / Sada su brojevi učitani, slijedi sortiranje */*
Postavi m=n
Postavi sortiran=false
Sve dok je sortiran=false čini

```
    Postavi sortiran=true
    Za svaki j=0 do m-2 učini slijedeće:
        Ako je v[j] > v[j+1] čini /* Treba zamijeniti v[j] i v[j+1] */
            Postavi sortiran=false
            Postavi x=v[j]
            Postavi v[j]=v[j+1]
            Postavi v[j+1]=x
            Kraj Ako je
        Kraj Za svaki j
    Smanji m za 1
    Kraj Sve dok
/* Sada su brojevi sortirani, slijedi ispis */
Za svaki i=0 do n-1 ispiši v[i]
```

Komentar rješenja:

Postupak redanja podataka po nekom kriteriju zove se sortiranje.
Prikazani algoritam je tzv. "Bubble sort".
Sve brojeve (ili općenito podatke) pohranimo (memoriramo) u matricu ili indeksiranu varijablu. Indeksirane varijable ili matrice predstavljaju skup podataka označen zajedničkim imenom. Pojedini podatak u skupu određen je imenom skupa i položajem u skupu ili indeksom. Koristićemo dogovor da je početni član skupa određen indeksom 0, pa prema tome indeksi u skupu od n elemenata poprimaju vrijednosti od 0 do n-1. Ako je skup razmješten u jednoj dimenziji, tada govorimo o jednodimenzionalnoj matrici ili vektoru. Ako je skup razmješten u dvodimenzionalnu tabelu, u retke i stupce, tada govorimo o matrici. U ovom primjeru koristimo vektor v od 999 članova. Svaki član adresira se indeksom. Na. pr. v[0] je prvi član skupa, v[j-1] je j-ti član i t. d. Potrebno je da na početku definiramo vektor od 999 elemenata. Zbog toga broj elemenata koji ćemo učitati i zatim sortirati ne može biti veći od 999 . Nakon učitanog broja n ispitujemo njegovu ispravnost te ako je n ispravan, učitavamo n brojeva u vektor v. Nako toga slijedi samo sortiranje. Ono se svodi na uspoređivanje j-tog i idućeg, j+1 broja u vektoru v. Ako je naredni broj manji od prethodnog (v[j+1] < v[j]), izvršićemo njihovu zamjenu i zabilježit pomoću logičke varijable "sortiran" da svi brojevi još nisu poredani (true i false su logičke konstante). Ovaj postupak ponavljamo tako dugo, dok sve brojeve ne poredamo i nakon prolaska kroz vektor dobijemo sortiran=true. Nakon svakog uspoređivanja još neporedanih brojeva u vektoru, njihov broj se smanjuje za 1. Zato broj m koji označava broj neporedanih podatama nakon toga smanjujemo za 1. Nakon obavljenog sortiranja (sortirani=true) ispisujemo podatke iz vektora v.

Primjer

Pretpostaviti da postoji funkcija Rnd(n), koja pri svakom svom pozivu daje slučajni broj koji je prirodan broj u intervalu 1 do n. Npr. ako naznačimo "Postavi i= Rnd(34)" tada će varijabla i poprimiti vrijednost nekog (slučajnog) cijelog broja između 1 i 34. Ponavljanjem ove naredbe, varijabla i će svaki puta poprimiti neku "slučajnu" vrijednost ali uvijek u istom intervalu (ako funkcija Rnd daje jednoliko raspoređene slučajne brojeve, tada će se pri vrlo velikom broju njenog pozivanja svaki broj pojaviti približno jednako mnogo puta). Razraditi logiku programa koji će koristeći Rnd funkciju simulirati izvlačenje lota (od brojeva 1 do 39 nasumce izvući 7).

Rješenje:

Definiraj vektor loto od 39 elemenata
Za svaki i=0 do 38 postavi loto[i]=i+1
Postavi n=38
Za svaki i=1 do 7 učini
    Postavi j= Rnd[n]
    Postavi k=loto[j]
    Ispiši izvučeni broj
    / k je "izvučeni" slučajni broj; on se ne može*
    ponovo izvući, pa ga treba izbaciti iz tabele:
    na njegovo mjesto stavljamo zadnji, n-ti broj/*
    Postavi loto[j]=loto[n]
    Smanji n za 1
    Kraj Za svaki i

Komentar rješenja:

U tabelu (vektor) loto stavili smo brojeve 1 do 39. S n=39 smo naznačili da u tabeli ima 39 brojeva. Iza toga započinjemo iteraciju koja će se obaviti 7 puta, jer ćemo izvući 7 brojeva. Nakon toga dobivamo slučajni broj j koji je između 0 i n-1 (prvi puta n=39). Broj k koji je "izvučen" nalazimo na j-tom mjestu tabele. Ispisujemo taj broj i sada ga izbacujemo iz tabele tako, da na njegovo mjesto stavljamo posljednji tj. n-1. broj. n smanjujemo za 1 jer se izbor brojeva za izvlačenje smanjio za 1.

Zadatak Razraditi logiku programa koji će simulirati bacanje kocke. Baciti kocku 6000 puta i naći i ispisati broj pojavljivanja svakog od brojeva 1 do 6. Koristiti Rnd funkciju kao u primjeru 1.5.

Zadatak Razraditi logiku programa koji će simulirati istovremeno bacanje dviju kocaka. Baciti kocke zadani n broj puta. Naći u koliko posto slučajeva su se na obje kocke pojavili isti brojevi.

Primjer

Razraditi logiku programa koji će koristiti dvije matrice a1 i a2 od 100 redaka i stupaca. Program najprije učita brojeve m i n koji označavaju broj redaka i broj stupaca koji će biti popunjeni u matricama. m i n moraju biti manji od 100. Ako taj uslov nije zadovoljen, ispisati poruku i ponovo učitati m i n. Učitati podatke (brojeve) u m redaka i n stupaca matrice a1 (ukupno m * n brojeva). Naći sumu svakog redka i svakog stupca matrice a1 i te sume staviti u posljednji redak (s indeksom 99) odnosno stupac matrice. Naći ukupnu sumu S svih elemenata matrice a1 i taj broj staviti u 100. redak i 100. stupac. Naći sve relativne podatke a[i][j]/S i njihove vrijednosti u postocima staviti u odgovarajuće redke i stupce matrice a2. Ispisati matricu a2.

Rješenje:

Definiraj matrice a1 i a2 od 100 redaka i stupaca.
Ponavljaj
    Učitaj m i n.
    Ako je m > 99 ili n > 99 ispiši poruku o pogrešno zadanom m ili n
    Sve dok je m>99 ili n>99
Za svaki i=0 do m-1 učini
    Za svaki j=0 do n-1 učitaj a1[i][j]
Za svaki i=0 do 99 učini slijedeće:
    Postavi a1[i][99]=0 / Postavi na 0 zadnji redak i stupac */*
    Postavi a1[99][i]=0
    Kraj Za svaki i
Postavi S=0
Za svaki i=0 do m-1 / Nađi sumu redka, sumu stupca i ukupnu sumu s */*
    Za svaki j=0 do n-1 učini
        Povećaj a1[i][99] za a1[i[j]
        Povećaj a1[99][j] za a1[i][j]
        Povećaj S za a1[i][j]
        Kraj Za svaki j
Postavi a1[99][99]=S
/ Relativne podatke stavi u matricu a2 i ispiši je */*
Za svaki i=0 do m-1
    Za svaki j=0 do n-1 učini
        *Postavi a2[i][j]=a1[i][j]/a1[99][99]*100*
        Ispiši a2[i][j]
        Kraj Za svaki j

Komentar rješenja:

Na početku smo definirali matrice koje ćemo koristiti u algoritmu. Kod izrade rješenja u programskom jeziku kakav je C, potrebno je definirati na početku programa sve varijable no za bolju čitljivost algoritma u njemu naznačujemo samo definicije vektora i matrica. Ako su m i n manji od 100, učitat ćemo podatke u m redaka i n stupaca matrice a1. Služimo se konvencijom, da prvi indeks matrice označava redak, a drugi indeks stupac matrice tj. a1[i][j] označava element u i-tom redu i j-tom stupcu. Nakon učitavanja postavljamo na nulu vrijednosti u zadnjem (s indeksom 99) retku i stupcu matrice a1. Iza toga "prolazimo" kroz sve retke i sve stupce matrice tako da za svaki redak i (od 0 do m-1) uzimamo element a1[i][j] u svakom j-tom stupcu (j ide od 0 do n-1) i povećavamo zadnji stupac svakog reda (a1[i][99]) i zadnji redak svakog stupca (a1 [99][j]) za svaki element matrice (a1[i][j]). Na isti način u zadnji redak i stupac (a1[99][99]) stavljamo ukupnu sumu. Relativni iznos u postotku dobivamo tako da svaki podatak iz matrice a1 dijelimo sa a[99][99] i množimo sa 100. Rezultat stavljamo u matricu a2. Tako dobiveni rezultat konačno ispisujemo u dvostrukoj iteraciji po i i j na kraju algoritma.

Zadatak Razraditi logiku programa koji će koristiti matricu od 20 redaka i 20 stupaca. Učitati brojeve u svaki redak i stupac te matrice. Naći i ispisati sumu članova matrice na njenoj glavnoj dijagonali (glavna dijagonala ide od gornjeg lijevog u donji desni ugao matrice).

Zadatak Razraditi logiku programa koji će koristiti matricu od 50 redaka i 50 stupaca. Učitati broj m koji mora biti manji ili jednak 50. Ako taj uslov nije zadovoljen ponoviti učitavanje broja m. Učitati m * m brojeva u m redaka i stupaca matrice. Naći i ispisati proizvod članova na sporednoj dijagonali matrice (sporedna dijagonala ide od desnog gornjeg u lijevi donji ugao).

Zadatak Razraditi logiku programa koji će koristiti matricu od 50 redaka i stupaca. Učitati brojeve m i n koji određuju broj popunjenih redaka (m) i stupaca (n) matrice. m i n moraju biti manji od 51. Ako taj uslov nije zadovoljen ponoviti učitavanje brojeva m i n. Učitati brojeve u m redaka i n stupaca matrice. Naći i ispisati najveći broj u svakom od m retka matrice.

TEHNIKE PROGRAMIRANJA

MODULARNO PROGRAMIRANJE - temelji se na raspodjeli programskih funkcija na manje NEOVISNE module između kojih se potrebna komunikacija ostvaruje prijenosom operanda. Modularna struktura omogućava timski rad jer se isti modul može koristiti u raznim drugim dijelovima programa ili u drugim programima. Na primjer modul za brisanje podataka u nekoj bazi podataka obrisaće podatke temeljem unesenih vrijednosti operanda kao rednih brojeva podataka koji se imaju obrisati. S drugim vrijednostima operanda modul će poslužiti za brisanje u nekoj drugoj bazi podataka iste strukture ali po nekom drugom kriteriju, recimo brisanje po nazivu a ne po rednom broju.

STRUKTUIRANO PROGRAMIRANJE - predstavlja način pisanja programa u kojem se poštuje strogi SLIJED odvijanja programa ili njegovih modula. Nema skokova u odvijanju programa prema početku ili kraju već se svaki smjer odvijanja programa određuje izborom DA-NE. Moduli su međusobno povezani hijerarhijski. Time se omogućava da se tok programa može bezprijekorno pratiti od početka do kraja što olakšava čitljivost i otklanjanje grešaka.

METODE strukturnog programiranja:

Deklariranje svih varijabli prije korištenja (naprimjer na početku programa),
Izbjegavanje nekih naredbi nestrukturnog programiranja (npr *goto*).
Tjeraju programere na programersku disciplinu i urednost, ograničavaju slobodu programera. PRIMJERI: Pascal, dBase, Ada.

INTERAKTIVNO PROGRAMIRANJE - je tehnika kada se za izradu programa koriste već gotove logičke strukture, izrađene na primjer od strane isporučitelja programskog alata, te ih programer dorađuje prema svojim potrebama i rezultate rada može odmah i testirati. Produktivnije je ali manje sistematično.

OBJEKTNO USMJERENO PROGRAMIRANJE - nastalo je kao odgovor na nemogućnost definiranja globalnih zajedničkih podataka u struktuiranom programiranju. Objektno-orijentirano programiranje prestavlja oblik programiranja kod kojeg programer osim tipa i strukture podataka definira i operacije (funkcije) koje su primjenjive na određenu strukturu. Na taj način podatkovna struktura postaje objekt koji uključuje i podatke i funkcije. Nadalje, programeri definiraju relacije između objekata (npr.neki objekt može naslijediti svojstva od drugog objekta).Naime, pojedini moduli programa koriste zajedničke podatke i umjesto da se podaci proslijeđuju kroz razine strukture do modula koji ih treba, omogućava se modulu da ih direktno dohvati bez obzira na razinu u kojoj se nalazi. Podaci i operacije nad tim podacima spajaju se u cjeline nazvane OBJEKT. Svi objekti iste vrste svrstavaju se u KLASU a svaki od njih ima zasebno stanje unutar klase. U klasi se definiraju zajedničke postavke za sve njene objekte koje oni mogu i ne moraju koristiti i veze prema drugim klasama. Povezanost klasa tvori hijerarhijsku strukturu programa.Kojim se od jezičkih procesora koristiti ovisi o vrsti problema koji se riješava te o mogućnostima i vrsti Digitalnog Računarskog Sistema na kojoj će se primjeniti. Stoga tu nema strogih

zakonitosti. Jenostavniji zadaci rješavaće se na PC-računarima s QBASIC, PASCAL, FORTRAN, C++ ili VISUAL BASIC jezičkim procesorom, a za knjigovodstvene svrhe ili obradu učeničke populacije koristiće se alati koji mogu upotrebiti neki od poslužitelja baze podataka kao DELPHI. Jedna od vrhunskih programskih podrški za praćenje rada velikih firmi i ustanova, LOTUS-NOTES, vrlo je kvalitetno ali i vrlo skupo rješenje.

SVOJSTVA Objektno orijentiranih programa

Objekt je zaseban program, koji je sposoban obraditi određenu zadaću. Sastoji se od apstraktnih podataka (engl. *abstract data types*) i postupaka (engl. *methods*).
Apstraktni podaci je naziv za podatke čiji je oblik nedostupan izvan objekta. Glavni program nema direktan pristup takvim podacima. Operacije s apstraktnim podacima mogu obavljati samo postupci. Objekt može biti napisan u programu različitom od onog u kome programer piše program. Programer ne zna ništa o tome kako je objekt napisan (zatvorenost, engl. *encapsulation*), nego treba znati samo propisani oblik slanja poruke objektu. Mogućnost nasljeđivanja (engl. *inheritance*) svojstava prije napisanog objekta, kako se ne bi morao pisati zajednički dio.
Višezadaćnost (engl. polymorphism) je mogućnost da se dobije odgovarajući rezultat pri slanju iste poruke različitim objektima.

PREDNOSTI: Pisanje programa pomoću objekata jednostavnije je i s gledišta preslikavanja stvarnog svijeta (npr poslovnog svijeta) u oblik razumljiv računaru, tj. u odnose među objektima.

NEDOSTACI: Zbog načina rada OOP u načelu se sporije izvršavaju od tradicionalnih programa,
navika programera na tradicionalno programiranje i mnogo gotovih starih programa.

Izbor višeg programskog jezika

Viši programski jezici opće namjene:

Jezici opće namjene su oni programski jezici koji nisu namijenjeni uskom području primjene, već je njima moguće rješavati širok raspon zadataka.
Postoji mnogo ovih jezika, a mi ćemo spomenuti samo one koji su značajni ili su u širokoj upotrebi:
Programski jezik **ADA** (ime po grofici Augusti Adi Loverance): (Jean Inchbiah 1979.) Temeljen je na Pascal-u, omogućava višezadaćni rad, direktan pristup sklopovskom dijelu računara, primjenu kod sistema s odzivom u stvarnom vrenenu (engl. *real time*) i dr.

Programski jezik **ALGOL** (engl. *algorithmic language*): (1957.-1960.) Namijenjen je obradi podataka pri naučnim istraživanjima i inženjerskim proračunima.

Programski jezik **APL** (engl. *a programing language*): (Kenneth Iverson 1957.-1967.) Namijenjen je sažetom i jednostavnom prikazu matematičkih algoritama, posebno kod vektorskog i matričnog računa

Programski jezik **BASIC** (engl. *beginner's all-purpose symbolic instruction code,* u prijevodu "jezik opće namjene za početnike"): (John G. Kemeny i Thomas E. Kurtz 1962.-1964.) Odigrao je veliku ulogu u približavanju računara i programskih postupaka ljudima koji nisu specijalisti za računare i omogućio je stručnjacima raznih struka rješavanje zadataka pomoću računara.

PREDNOSTI: jednostavnost, razumljivost, rasprostranjenost gotovo na svim vrstama računara.
NEDOSTACI: nestrukturiranost jezika, pa je temeljni prigovor sadržan u izjavi E. Dijkstra: "Studenti koji su bili prvo izloženi BASIC-u su beznadno mentalno osakaćeni kao potencijalni programeri."

Popularne INAČICE: True BASIC, MS Quick BASIC, MS Visual BASIC.

Programski jezik **C** (prvobitno razvijeni CPL, BCPL, B): (Dennis Ritchie 1972.) Najpopularniji je programski jezik opće namjene. Razvijen je uporedo s operativnim sistemom Unix, koji je i napisan u C-u. Vrlo je složen i velikih mogućnosti, a namijenjen je ponajprije stručnjacima koji se profesionalno bave pisanjem programa.
Ima dobra SVOJSTVA viših programskih jezika, ali i mogućnost manipulacije sklopovskim dijelovima računara na niskoj razini.
Popularne INAČICE: Microsoft C, Borland Turbo C, Watcom C, nekoliko objektno orijentiranih inačica C++.

Programski jezik **COBOL** (engl. *common business oriented language,* u prijevodu "opći poslovno orijentirani jezik"): (grupa stručnjaka u odboru konferencije CODASYL potkraj 50.-tih g. 20.st.) Jedan je od prvih viših programskih jezika. Namijenjen je poslovanju u bankama, obračunima preduzeća, obračunima plaća i sl.

Programski jezik **Forth**: (Charles H. Moore kasnih 60.-tih 20.st., smatrao ga jezikom četvrte generacije, engl. *fourth generation language*) Ima mogućnost jednostavnog definiranja novih naredbi na temelju postojećih. Programer može jednostavno smisliti riječ kojoj će pridijeliti željenu funkciju, koja nakon toga postaje naredba ovog jezika. Ima ugrađen skup naredni za upravljanje prividnom memorijom, za direktan pristup ulazno-izlaznim jedinicama i sl. Programi se brzo izvršavaju.

Programski jezik **FORTRAN** (skraćenica od engl. *formula translation*, u prijevodu "prevoditelj formula"): (John Backus 1954.-1958.) Najstariji je viši programski jezik. Namijenjen je ponajprije tehničkoj i naučnoj upotrebi, gdje je potrebno obavljati raznovrsne računske operacije.
INAČICE: FORTRAN II, FORTRAN IV, FORTRAN 66, FORTRAN 77, FORTRAN 90, Microsoft FORTRAN, Lahey Computer Systems Personal FORTRAN, Microway FORTRAN.

Programski jezik **LOGO**: (Wallyy Fuerzeig i Seymour Papert 1966.-1968.) Namijenjen je djeci i ostalim početnicima na računaru. Prvi je programski jezik koji je omogućavao jednostavno crtanje pomoću računara bez složenog programiranja.

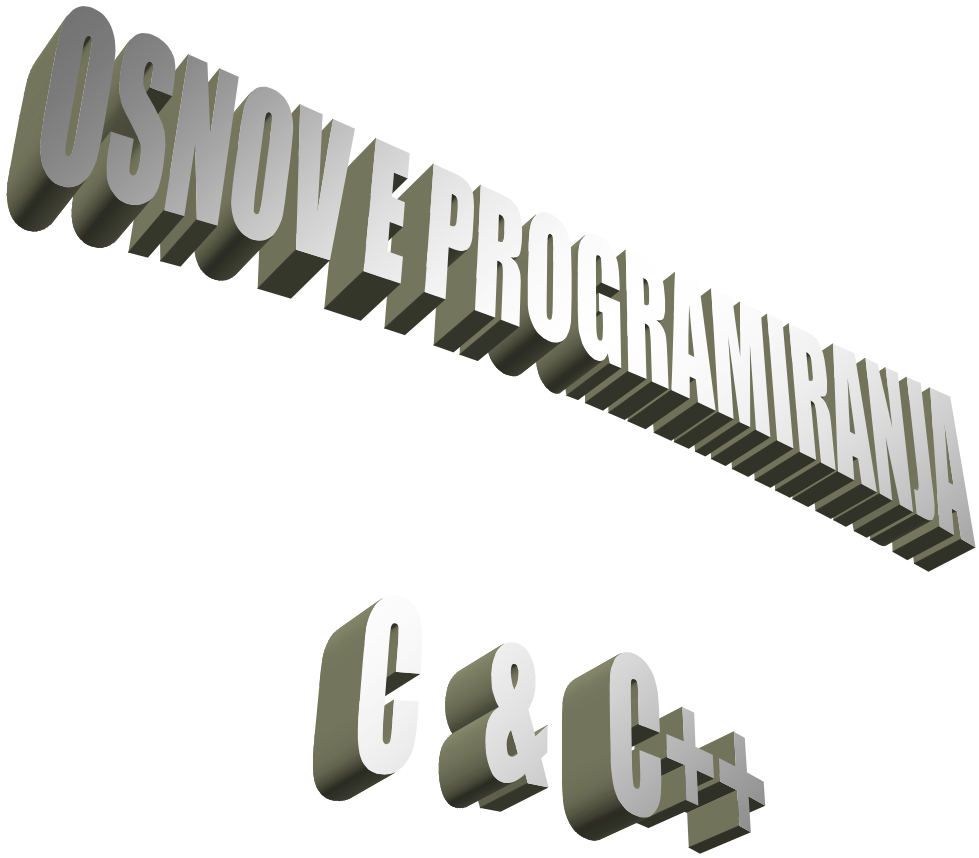
Programski jezik **Modula-2** (skraćenica od engl. *modular language,* u prijevodu "modularni jezik"): (Niklaus Wirth 1977.) Glavna mu je odlika modularnost, višezadaćni rad i mogućnost prevođenja modula (dijelova programa). Smatra se pretećom suvremenih objektno orijentiranih jezika.

Programski jezik **PL/M** (engl. *programing language for microprocessors*): (Gery Kildall 1972.) Namijenjen je ponajprije primjeni kod mikroprocesora firme Intel. Originalna mu je namjena bila poslužiti kao programski jezik za CP/M operacijski sistem. Nastao na PL/I, koji je trebao objediniti najbolja svojstva FORTRAN-a, COBOL-a i ALGOL-a.

Programski jezik **PASCAL** (dobio ime po Blaise Pascalu): (Niklaus Wirth 1967.-1971.) Posebno omogućuje strukturno programiranje, koje olakšava pisanje, održavanje i popravak programa. Prvobitno je bio namijenjen učenju strukturnog programiranja, ali je stekao veliku popularnost u akademskim krugovima. Danas ga koriste neprofesionalci kao "razbibrigu", ali i za profesionalne primjene. Zasnovan je na ALGOL-u, a po mogućnostima je između BASIC-a i C-a.
Popularne INAČICE za PC: Turbo Pascal, Borland Pascal, MetaWare Pascal, IBM Pascal, Object Pascal for Mac, UCSD Pascal.

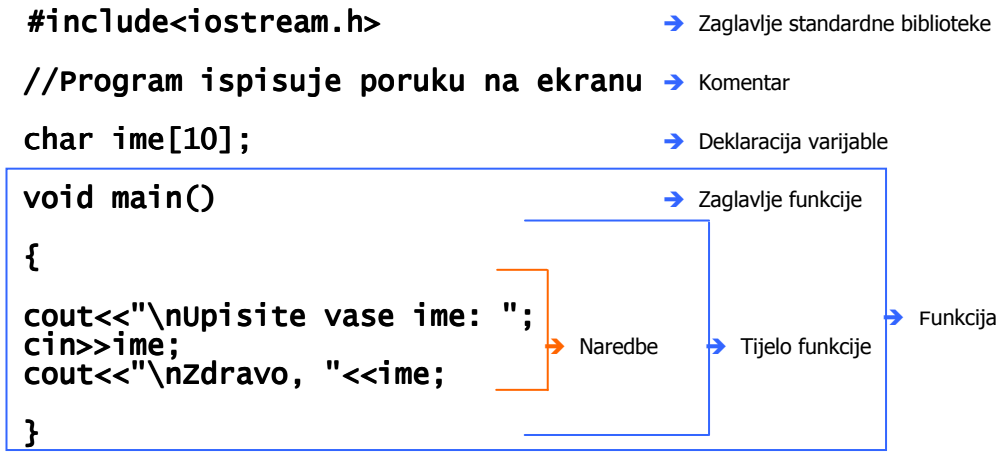
IZBOR PROGRAMSKOG JEZIKA :

Tip programa	***Najbolji jezici***	***Najlošiji jezici***
Strukturirani programi	*Ada, C/C++, Pascal*	*Asembler, Basic*
Kratki i nekvalitetni projekti	*Basic*	*Pascal,Ada, Asembler*
Brzo izvođenje programa	*Asembler, C*	*Interpreter poput Basic-a*
Matematički programi	*Fortran*	*Pascal*
Pogrami jednostavni za nadgradnju	*Pascal, Ada*	*C, Fortran*
Dinamička upotreba memorije	*C, Pascal*	*Basic*
Za okružja s ograničenom memorijom	*Basic, Asembler, C*	*Fortran*
Programi za rad u realnom vremenu	*Ada, Asembler, C*	*Basic, Fortran*
Upravljanje nizovima znakova	*Basic, Pascal*	*C*


OSNOVE PROGRAMIRANJA
C & C++

OSNOVNI POJMOVI

Osnovni elementi C++ programa



Ovaj kratki program ima nekoliko karakterističnih elemenata:

➔ **#include<iostream.h>**

#include - je uputa (direktiva) kompajleru da u program uključi tzv. zaglavlje biblioteke sa imenom "**iostream.h".**

Znak # (eng. hash) je signal za preprocesor. Svaki put kad se pokrene kompajler, prije kompajlera pokrene se i preprocesor. Preprocesor čita kod, odnosno program, tražeći linije koje počinju sa znakom #, i prevodi ove linije koda u specijalne naredbe spremne za kompajler.

Include - je instrukcija preprocesora koja se može interpretirati kao "*Ovo što slijedi je ime biblioteke. Nađi tu biblioteku i pročitaj je tačno ovdje.*" Uglaste zagrade **(< >)** oko imena biblioteke govore preprocesoru da ovu biblioteku potraži na uobičajenim mjestima, obično je to direktorij u kome se čuvaju svi *.h fajlovi. Prema tome, **#include<iostream.h>** ***znači, cijeli sadržaj fajla "iostream.h" umetni u ovaj program kao da je tu i napisan.***

Svi objekti u C++ jeziku grupisani su u biblioteke, i na početku svakog programa obavezno se uključuju sve biblioteke koje sadrže objekte koje koristimo u programu. Biblioteka **iostream.h** (skraćenica od eng**. input-output-stream**), između ostalog definiše objekte **"cout" i "cin"** koji koristimo za pristup ekranu. Ukoliko pišemo matematički program, na početku programa ćemo uključiti i biblioteku **math.h**, a ako pišemo program koji radi sa grafikom tada moramo uključiti i biblioteku **graphics.h**. Ideja svega ovoga je da se program ne opterećuje bespotrebno objektima koji se neće koristiti unutar programa. Nepotrebno je program opterećivati grafičkim objektima ako program neće koristiti grafiku.

Primjer: C++ biblioteke

```
#include<iostream.h>
#include<math.h>

int main()

{
int a,n,rez;
cout<<"\nBaza: ";
cin>>a;
cout<<"\nStepen: ";
cin>>n;
rez=pow(a,n);
cout<<"\n----------";
cout<<"\n"<a<<"^"<<n<<" = "<<rez<<endl;
return 0;
}
```

```
#include<iostream.h>
#include<graphics.h>

void main()
{
cout<<"Hello!";
}
```

Kada pišemo neki program, sasvim je jasno šta pokušavamo da uradimo. Međutim, za recimo mjesec dana, kad se vratimo istom programu, pojedini dijelovi koda mogu biti potpuno nejasni i konfuzni. Nije jasno kako dolazi do ove konfuzije, ali je sasvim jasno da uvijek dolazi. Da bi se oslobodili konfuzije i da bi pomogli drugima da razumiju naš kod, trebali bi koristiti komentare. Komentar treba biti kratak i jasan. Komentari su jednostavan tekst, ignorisan od strane kompajlera, koji informiše onoga ko čita naš program o čemu se radi u pojedinim dijelovima programa.

Komentari u C++ jeziku pojavljuju se u dva oblika:

// (eng. **double-slash**) komentar govori kompajleru da potpuno ignoriše sve od pojave znaka **"//"** do kraja tekućeg reda.

/* (eng. **slash-star**) komentar govori kompajleru da potpuno ignoriše sve što slijedi iza **"/*" znaka, pa sve do "*/"** (eng. star-slash) znaka. Svaki "/*" zahtjeva i "*/".

Primjer: Komentari

```
#include<iostream.h>

int main()
{

/* ovo je komentar
koji se zavrsava tek kad naidjemo
na star-slash znak */

cout<<"\nHello!\n";

// ovaj komentar zavrsava na kraju linije (reda)
// bas kao i ovaj komentar

cout<<"Ovaj komentar se zavrsio!\n";

// double-slash komentari mogu biti samo u jednoj liniji
/* kao i slash-star komentari */

return 0;

}
```

➔ char ime[10];

U C++ programskom jeziku varijabla je mjesto gdje se čuva informacija. Varijabla je lokacija u memoriji računara u kojoj se čuva (pohranjuje) neka vrijednost i iz koje je naknadno moguće uzeti tu vrijednost.

➔ void main()

Svaki C++ program mora imati najmanje jednu funkciju, odnosno funkciju koja se zove **main**. Tipične funkcije se pozivaju u toku izvršavanja programa. Program se izvršava liniju po liniju, upravo onako kako i izgleda sam izvorni kod, sve dok ne dođe do instrukcije koja poziva funkciju. Tada program prestaje sa izvršavanjem daljeg koda, odlazi na liniju gdje počinje data funkcija i izvršava je. Kada je izvršena funkcija, program se vraća na sljedeću liniju koda koja slijedi odmah iza linije koja je predstavljala poziv datoj funkciji.

Primjer: Tok izvršavanje programa koji poziva funkciju

```
1:      #include<iostream.h>
2:
3:      //Funkcija Ispis
4:      //ispisuje poruku na ekranu
5:      void Ispis()
6:      {
7:              cout<<"\nJa sam u Ispis funkciji\n";
8:      }
9:
10:     //Funkcija main – ispisuje poruku na ekranu,
11:     //a zatim poziva funkciju Ispis, i onda ispisuje
12:     //sljedecu poruku.
13:     int main()
14:     {
15:             cout<<"\nJa sam u main funkciji\n";
16:             Ispis();
17:             cout<<"\nPonovo u main funkciji\n";
18:             return 0;
19:     }
```

Funkcija *Ispis()* je definisana na linijama *5-8*. Kad je pozvana, funkcija Ispis() ispisuje poruku na ekranu. Linija 13 je početak programa. Na liniji 15, funkcija main() ispisuje poruku koja govori "*Ja sam u main funkciji*". Poslije ispisa poruke, linija 16 poziva funkciju Ispis(). Ovim pozivom počinje izvršavanje naredbi u funkciji Ispis(). U ovom slučaju cijela funkcija sastoji od samo jedne naredbe na liniji 7, koja ispisuje sljedeću poruku na ekranu "**Ja sam u Ispis funkciji**". Kada se završi Ispis() funkcija (linija 8), program se vraća na liniju koja slijedi odmah iza linije s koje je pozvana data funkcija tj. Ispis () funkcija, odnosno program se vraća na liniju 17, na kojoj se nalazi naredba koja ispisuje poruku na ekranu "Ponovo u main funkciji".

U programskim jezicima funkcija je obično potprogram koji vraća neku vrijednost ili rezultat, mada imamo i funkcije koje ne vraćaju nikakvu vrijednost, odnosno funkcije koje vraćaju void (**void znači ništa**). Funkcija koja sabira dva cijela broja može da vrati zbir ta dva broja, i prema tome može biti deklarisana kao funkcija koja vraća cjelobrojnu vrijednost (int). Funkcija koja samo ispisuje poruku na ekranu nema vratiti nikakvu vrijednost i zbog toga treba biti deklarisana kao void. ***Funkcija se sastoji od zaglavlja i tijela. Zaglavlje funkcije definiše vrijednost koju funkcija vraća, ime funkcije, kao i parametre koje funkcija očekuje.*** Prema tome, ako funkcija treba da sabere dva broja, parametri funkcije bi trebali biti ti brojevi.

Primjer: Zaglavlje funkcije

```
void ispisi(void)
{
cout<<"....;
...;
}
```

```
int zbir(int x, int y)
{
...;
...;
return (x+y);
}
```

Nakon zaglavlja funkcije slijedi **tijelo funkcije**. Tijelo funkcije počinje znakom "**{**", a završava znakom "**}**". Između vitičastih zagrada nalaze se nijedna ili više naredbi, odnosno blok naredbi i taj blok naredbi čini tijelo funkcije.

Naredbe određuju ili definišu redoslijed izvršavanja programa, izračunavaju vrijednosti ili ne rade ništa (null statement). Svaka naredba u C++ završava sa znakom **";"**, pa čak i null naredba, koja je samo **";"** i ništa više.

➔ **cout<<"\nUnesite vase ime: ";**

"**cout**" (skraćenica od eng. ***console out***, čita se [si aut]) predstavlja objekat tzv. izlaznog toka podataka (eng. *output stream*), koji je povezan sa standardnim uređajem za ispis (obično ekran). Znak "**<<**" predstavlja operator redirekcije (eng. redirection) odnosno umetanja (eng. insertion), koji pojednostavljeno možemo čitati kao "šalji na". Tekst "Unesite vase ime: " koji se nalazi između navodnika, predstavlja niz znakova koji šaljemo na izlazni tok. Dakle, gornja naredba se može interpretirati kao:

Pošalji niz znakova "Unesite vase ime: " na ekran.

➔ **cin>>ime;**

"**cin**" (skraćenica od eng. ***console in***, čita se [si in]) predstavlja objekat tzv. ulaznog toka podataka (eng. *input stream*), koji je povezan sa standardnim uređajem za unos (obično tastaturom). Znak "**>>**" predstavlja takođe operator redirekcije, ali suprotnog smjera od operatora "<<". Naziva se i operator izdvajanja (eng. Extraction), koji pojednostavljeno možemo čitati kao "šalji u", samo što je smisao desne i lijeve strane obrnut u odnosu na operator "<<". Šta se zapravo dešava? Po nailasku na objekat cin, program privremeno prekida rad, i omogućava nam da unesemo neki niz znakova preko tastature, sve dok ne pritisnemo tipku ENTER. Taj niz se čuva negdje u memoriji. Po nailasku na operator ">>", iz unesenog niza znakova izdvajaju se znaci do prvog razmaka (ili do kraja reda), i vrijednost izdvojenog niza smješta se u promjenljivu čije ime se nalazi desno od operatora ">>". Dakle, gornja naredba može se interpretirati kao:

Pošalji unos sa tastature u promjenljivu nazvanu "ime".

C – strukturni jezik

Uvod u programiranje u C-u

Elementi jezika dati su preko primjera programa, koji su po konceptu zajednički za bilo koji C program.

Prvi C program

```
/* Program #1   My first C program */
#include <stdio.h>
main()
{
printf("This is my first C program");
}
```

Linija komentara – počinje s /* … a završava */, sve što je između kompajler ignorira
Header files – sadrži podatke koji su potrebni programu
STDIO.H – mora biti u radnom okruženju programa koji se kompilira, sadrži ulazno izlazne funkcije
main() – funkcija, blok, koji svaki C program mora imati, C program počinje pozivom main() i završava kada se iz maina vraća
printf(“ …”) – ispisuje poruku na ekranu, u C-jeziku poruka se ispisuje pozivom standardne funkcije printf(), koja ispisuje sve što piše između navodika
Kraj naredbe – točka zarez (;)
Početak i kraj bloka – { … }

Pridruživanje vrijednosti varijabli

Program kreira varijablu value, daje joj vrijednost 1023, i nakon toga je ispisuje na ekran.

```
/* Program #2 */
#include <stdio.h>
main()
{
int value;
value = 1023;
printf("This program prinnts the value %d", value);
}
```

int value – deklarira varijablu kao cjelobrojnu
= - znak pridruživanja u C jeziku
printf() – sadrži dva argumenta odvojena zarezom
prvi – sadrži tekst i format – počinje sa **%** i upozorava da će se ispisati simbol koji nije karakter **%d – integer**
drugi – ime varijable koje se ispisuje (ispisuje se na mjestu formata)

Rješenje programa je tekst : This program prinnts the value 1023

Primjer 3

Program pretvara galone u litre, 1galon = 3.7854
Pošto se koriste cijeli brojevi uzeće se *1 galon = 4 litre*

```
/* This program converts gallons to liters */
#include <stdio.h>
main()
{
int gallons, litters;
printf("Enter number of gallons : ");
scanf("%d", & gallons);
litters = gallons * 4;
printf("%d liters", litters);
}
```

dvije cjelobrojne varijable – odvojiti imena zarezom
scanf() – standardna funkcija za čitanj vrijednosti sa tastature
prvi argument sadrži samo format %d – čita cjelobrojnu vrijednost
drugi &gallons – adresni operator da bi scanf dobro radila

Novi tip podatka

Program pretvara galone u litre uz upotrebu floating-point brojeva.(realni brojevi)

```
/* This program converts gallons to litters using floating point numbers */
#include <stdio.h>
main()
{
float gallons, litters;
printf("Enter number of gallons: ");
scanf("%f", & gallons);
litters = gallons * 3.7854;
printf("%f litters", litters);
}
```

float – ključna riječ za deklaraciju real varijabli s pomičnom točkom

%f – format u printf() i scanf() što upućuje funkcije na očekivanje floating-point podatka, ako se u programu napiše broj s decimalnom točkom, tada je to konstanta s pomičnom točkom

Zaključak

- C program mora imati funkciju main()
- Sve varijable valja deklarirati prije upotrebe
- C razlikuje više tipova podataka – int, float
- printf() funkcija se koristi za ispis na ekran
- scanf() funkcija se koristi za učitavanje s tastature
- Program se završava s pojavom znaka za kraj funkcije main()

Funkcije

C program je skup jedne ili više funkcija. Da bi napisali program prvo se napišu funkcije, a nakon toga se slože zajedno. Funkcija u C-u je potprogram koji sadrži jednu ili više C naredbi koje izvršavaju jedan ili više zadataka (u dobro napisanom C programu jedna funkcija – jedan zadatak). Svaka funkcija ima ime koje se koristi za poziv. Ime se daje proizvoljno, ali ime main() je rezervirano kao i ključne riječi (koje se koriste u jeziku kao naredbe). Ne može se koristiti funkcija unutar funkcije. Ime funkcije pa slijedi zagrada, da bi se funkcija razlikovala od varijable. main(), printf(), scanf()

Program je skup funkcija koje sam pišeš uključujući one iz C biblioteke.

Opći oblik funkcije u C-u

funkcija_ime(lista podataka)
{
… tijelo funkcije
}

Funkcija u C-u može vraćati vrijednost u pozivnu rutinu. Različiti su tipovi vrijednosti koje funkcija vraća. Ako nije deklariran tip vrijednosti koju funkcija vraća C kompiler predpostavlja da je vraća na vrijednost cijelobrojne. Svaka funkcija mora imati ime. Za poziv funkcije jednostavno se upotrijebi ime. List parametara daje imena i tipove varijabli kojima se pridružuju vrijednosti u funkciji. Ako parametara nema, zagrada je prazna.

Tijelo funkcije – sadrži naredbe C-a koje definiraju što funkcija radi.

} – oznaka kraja funkcije i povratka u pozivnu proceduru.

Program s dvije funkcije

```
/* This program contains two functions: main() and myfunc(). */
#include <stdio.h>
myfunc();
void main()
{ printf("in main()");
myfunc();
printf("back in main()"); }
myfunc()
{ printf(" inside myfunc() "); }
```

Program sadrži dvije funkcije **main()** i **myfunc().**
Program radi na sljedeći način. Prvo main() počinje izvođenje i zove printf() naredbu. Nakon toga main() zove myfunc(). Poziv se postiže ispisom imena myfunc(); poziv funkcije je C naredba te se mora izvršiti. Nakon toga myfunc() zove printf() i vraća se u main() kod zatvorene vitičaste zagrade. Nakog toga main() poziva printf() drugi put.

Izlaz na ekranu je: *in main() inside myfunc() back in main()*

Argumenti funkcije

Argument – vrijednost za koju se računa funkcija.
Formalni parametar – poprima vrijednost argumenta.

Funkcija printf() ima argument, sting koji se ispisuje na ekranu (niz znakova). Funkcija u C-u može imati više ili nijedan argument (gornji limit može biti 31).

Kada se funkcija definira, varijabla koja će poprimiti vrijednost argumenta mora se deklarirati. To su formalni argumenti – parametri funkcije. Funkcija navedena dolje ispisuje proizvod dva cjelobrojna argumenta koji se koriste u pozivu.

```
mul(int x, int y)
{
printf(“%d”, x * y);
}
```

Svaki put kad se mul() zove, pomnoži se vrijednost pridruženu x s vrijednošću pridruženu y. *Zapamtiti: x i y poprime vrijednost u trenutku poziva.*

Dat je primjer poziva funkcije mul()

```
/* A simple program that demonstrates mul(). */
#include <stdio.h>
mul(int x, int y);
void main()
{
mul(10, 20);
mul(5, 6);
mul(8, 9);
}
mul(int x, int y)
{
printf("%d ", x * y);
}
```

Program će ispisati 200, 30, 72 ekranu. Kada se mul() zove prvi put 10 se kopira u x, a 20 u y. Kod drugog poziva 5 se kopira u x, a 6 u y. U trećem pozivu 8 se kopira u x, a 9 u y. Argument je vrijednost za koju se funkcija računa, definira se pozivom, a formalni parametar je vrijednost u koju se argument kopira (pridružuje).

Funkcija vraća vrijednost

Funkcija može vratiti vrijednost u pozivnu rutinu. U C-u se vrijednost vraća naredbom **return.**

Opći oblik naredbe: **return value;** **(** value – se vraća **)**

Program piše proizvod dva broja. Vrijednost je pridružena varijabli, tako da se funkcija piše na desnu stranu naredbe pridruživanja.

```
/* A program that uses return. */
#include <stdio.h>
mul(int x, int y);
void main()
{
int answer;
answer = mul(10, 11); /* assign return value */
printf("The answer is %d \n", answer);
}
/* this function returns a value. */
mul(int x, int y)
{
return x * y;
}
```

U ovom programu mul() vraća vrijednost x*y uz pomoć naredbe **return.** Ova vrijednost je pridružena varijabli answer. Dakle vrijednost vraćena return naredbom postaje vrijednost mul() u pozivnoj rutini.

int : vrijenost – vraćena po default-u.

main() function

Funkcija **main()** označava početak programa, te je zgodno da se piše na početku. Samo je jedna main() funkcija u programu, ako ih ima više program neće znati gdje da počne. Većina kompilera javlja pogrešku.

printf() funkcija

Opći oblik: ***printf("kontrol string",argument list)***

Kontrolni string sadrži znakove koji se ispisuju na ekranu ili naredbe formata koje se odnose na listu argumenata ili oboje. **%d** – za cijele brojeve i **%f** – za pomičnu točku. Format se može naći bilo gdje u kontrol stringu. Formati i argumenti pridružuju se od lijeva na desno. Broj formata u kontrol stringu govori funkciji printf() koliko argumenata da očekuje. Ako se piše jedan znak, koristi se format **%c**.
printf("%c %c %c", 'A', 'B', 'C');

Primjeri:

printf(“this is a string %d”, 100); ***displays:*** *this is a string 100*
printf(“%d is decimal,%f is float”,10,110.789); ***displays:*** *10 is decimal, 110.789 is* loat
printf(“this is %c in uppercase %c”, ‘a’, ‘A’); ***displays****: this is a in uppercase A*

\n

Postoji mogućnost **ispisa u novi red**. Za to se koristi format \n.

```
/* This program demonstrates the \n code which generates a new line */
#include <stdio.h>
main()
{
printf("one\n");
printf("two\n");
printf("three");
printf("four");
}
```

Program daje na ekranu:
one
two
threefour

Format \n može se pisati bilo gdje u kontrol stringu, a ne samo na kraju.

scanf() funkcija

Opći oblik: ***scanf(“kontrol string”,argument list);***

Pretpostavićemo da kontrol string može samo sadržavati format kod. %d i %f koji kažu da učita cijeli broj, odnosno broj s pomičnom tačkom. Broj argumenata i broj formata mora biti isti.

Identifikatori u C jeziku

Identifikator u C-u je ime funkcije, varijable ili bilo koji, od korisnika, definiran član. Može biti proizvoljno dugačak. Ime varijable počinj slovom, može slijediti donja crtica za pomoć u čitljivosti varijable. Mala slova i velika slova su različita imena (count ≠ Count).
Ključne riječi ne možemo koristiti za identifikator.

Naredbe

If if(uslov) naredba ; uslov – istina ili laž (istina ≠ 0, laž = 0)

```
/* This program ilustrates the if statement */
#include <stdio.h>
void main()
{
int a, b;
printf("Enter first number: ");
scanf("%d ", & a);
printf("Enter second number: ");
scanf("%d", & b);
if(a < b) printf("First number is less than second");
}
```

Naredba se izvršava smo ako je uslov istina. *if(10<11) printf(“10 je manje od 11”);*

for petlja

Opći oblik: **for(inicijalizacija, uslov, korak) naredba;**

Inicijalizacija – postavlja kontrolnu varijablu petlje na početnu vrijednost.
Uslov – tesetira se kod svakog ponavljanja petlje. (ne nula) – petlja se ne nastavlja.
Korak – porast kontrolne varijable.

Primjer: Program ispisuje brojeve 1 do 100 na zaslon.

```
/* A program that ilustrates the for loop */
#include <stdio.h>
main()
{
int count;
for(count=1; count<=100; count=count+1)
printf("%d", count);
}
```

ključne riječi

Ključne riječi moraju se pisati malim slovima (RETURN ≠ return).

VARIJABLE, KONSTANTE I OPERATERI

Osnovni podaci u C-u

Tip:	Broj bita:
char	8
int	16
float	32
double	64
void	0

Deklaracija varijabli

Int i, j, k;
char ch, chr;
float f, balance;
double d;

Lokalne varijable

Deklariraju se unutar funkcije.

Primjer:

```
#include <stdio.h>
func();
void main()
{
int x;
x = 10;
func();
printf("%d", x);
}
func()
{
int x;
x = -199;
printf("%d\n", x);
}
```

U C-u lokalne varijable se kreiraju kad se funkcija zove, a na izlazu iz funkcije se gube.

Formalni parametri

Deklaracija se obavlja unutar zagrade, koja se piše u definicionoj naredbi funkcije.

Globalne varijable,

Deklariraju se na početku programa, prije main() funkcije. Vrijednost je poznata u cijelom programu.

Primjer:

```
#include <stdio.h>
int count;
func1();
func2();
void main()
{
int i;
for (i=0; i<10; i++)
func1();
}
func1()
{
printf("count: %d", count);
func2();
12
}
func2()
{
int count;
}
```

Ako globalna i lokalna varijabla imaju isto ime tada se unutar funkcije gdje je lokalna varijabla deklarirana pojavljuje lokalno, a ne utiče na globalnu varijablu.

Modifikatori

Osim za tip void, C dopušta osnovnim podacima da se pišu nakon modifikatora.

signed
unsigned
long
short

Primjer:

```
/* Program pokazuje razliku signed i unsigned */
#include <stdio.h>
void main()
{
int i; /* signed integer */
unsigned int j; /* unsigned integer */
j = 60000;
i = j;
printf("%d %u", i, j);
}
```

Primjer:

```
#include <stdio.h>
main()
{
char letter;
for(letter='Z'; letter>='A'; letter--)
printf("%c", letter);
}
```

Konstante

Može biti bilo koji podatak osnovnog tipa, uz uporabu sufiksa iza vrijednosti:

“F” – float
“L” – long duble
“U” – unsigned int

Oktalne i hexadecimalne konstante

(10) 8 → 0, 1, 2, 3, 4, 5, 6, 7
(10) 16 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
int hex = 0 x FF /* 255 */ nulu slijedi x
int oct = 011 /* 9 */ počinje nulom

String konstante

Niz znakova. Piše se između dvostrukih navodnih znakova.
%s – format kod za string konstantu

Primjer:

```
#include <stdio.h>
main()
{
printf("%s %s %s %s", "this", "is", "a", "test");
}
```

Backslash character konstanta

\n novi red	**\b** backspace	**\r** carriage return
\t tab	**\”** dvostruki navodnik	**\’** jednostruki navodnik
\0 nula	**** backslash	**\¡** vertikalni tabulator
\a alarm	**\N** oktalna konstanta	

\xN heksadecimalna konstanta
\b pomiče kursor u lijevo za jedan znak
\f ide na početak nove stranice (form feed)
\r ide na početak tekuče linije (carriage return)

Inicijalizacija varijabli

tip varijable_ime = konstanta;

char ch = ‘a’ ; int prvi = 0 ; float bla = 123.23F ;

Globalne varijable – na početku programa
Lokalne varijable – kod svakog poziva funkcije u kojoj se deklariraju (ne zna se vrijednost do prvog pridruživanja ako nije inicijalizirana)

Primer: Zbir svih brojeva od 1 do učitanog broja.

```
#include <stdio.h>
total(int x);
void main()
{
int t;
printf("ENTER A NUMBER: ");
scanf("%d", & t);
total(t);
}
total(int x)
{
int sum = 0; /* iniijalizacija */
int i, count;
for(i=0; i<x; i++){
sum = sum + 1;
for(count=0; count<10; count++) printf(".");
printf(" THE CURENT SUM IS %d \n", sum);
}
}
```

Operatori

Operatori kažu kompajleru da obavi neke matematičke ili logičke operacije.
Aritmetički operatori: - , + , * , / , % - daje ostatak cjelobrojnog djeljenja
-- , ++ , Modul (%) operator ne može se rabiti za float i double

Primjer:

```
#include <stdio.h>
main()
{
int x, y;
x = 10;
y = 3;
printf("%d", x/y); /* 3 */
printf("\n%d", x%y); /* 1 */
x = 1;
y = 2;
printf("\n%d %d", x/y, x%y); /* 0 1 */ }
```

OPERATOR SHORT EKVIVALENT

+ = x + = 2 ; x = x + 2 ; - = x - = 2 ; x = x – 2 ; * = x * = 2 ; x = x * 2 ;
/ = x / = 2 ; x = x / 2 ; % = x % = 2 ; x = x % 2 ;

Increnent, dekrement – brže se izvršava od pridruživanja te se korist kad je moguće

x = x + 1 ; ++ x ; x ++ ; x = x – 1 ; -- x ; x -- ; x = 10 ; /* 10 */
y = ++ x ; /* 11 */ x = 10 ; /* 10 */ y = x ++ ; /* 10 */

Bit operatori

& I **|** ili **^** ekskluzivni ili **~** komplement
<< shift lijevo **>>** shift desno EKSKLUZIVNI ILI (xor)

Relacioni i logički operateri

> && AND >= || OR < ! NOT <= == !=

Izrazi

Konstante, varijable, operateri i zagrade čine izraz.

Primjer:

```
#include <stdio.h>
main() /* print i and i/2 */
{
int i;
for(i=1; i<=100; ++i)
printf("%d/2 is: %f\n", i, (float)i/2);
}
```

Cast – osigurava ispis decimala, bez toga bi računar izvršio cjelobrojno djeljenje.

NAREDBE SELEKCIJE, ITERACIJE I SKOKA

Selekcija : **if , switch**

Iteracija : **for , while , do-while**

Skok : **break , continue , return , goto**

if naredba

Opći oblik:

```
if ( uslov ) naredba;
else naredba;
```

```
if ( uslov )
{
blok 1
}
else
{
blok 2
}
```

Primjer:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
int magic;
int guess;
magic = rand(); /* get a random number */
printf(" Enter your guess: ");
scanf("%d", & guess);
if(guess==magic)printf("\n **Right** ");
else printf("\n ...sorry, you are wrong... ");
}
```

Uslovni izraz

Izraz koji za rezultat daje 0 ili ≠0 (laž ili istina).

Primjer: Program čita dva broja sa tastature, ispisuje kvocijent. Kontrola djeljenja s nulom provedena je drugim brojem.

```
#include <stdio.h>
main()
{
int a, b;
printf(" Enter two numbers: ");
scanf("%d %d", &a, &b);
if(b) printf("%d\n", a/b); /* if(b!=0) printf("%d", a/b); - ne koristi se ovaj oblik */
else printf(" Cannot devide by zero\n");
}
```

for petlja

Opći oblik : for (inicijalizacija; uslov; increment) naredba;

ili cijeli blok
```
for ( inicijalizacija; uslov; korak )
{
blok naredbi
}
```

Petlja se vrti sve dok je uslov istina, kad uslov postane laž izvršava se naredba koja sljedi (for blok).

Inicijalizacija – naredba pridruživanja, postavlja vrijednost kontrolne varijable.
Uslov – određuje kad će se izaći iz petlje.
Increment – definir kako se kontrolna varijabla mjenja pri svakom ponavljanju petlje.

Primjer.

Program ispisuje vrijednosti drugog korijena brojeva od 1 do 100.

```
#include <stdio.h>
#include <math.h>
main()
{
int num;
double sq_root;
for(num=1; num<100; num++){
sq_root = sqrt((double)num);
printf("%d %f \n", num, sq_root);
}
}
```

Primjer.

```
Ispis brojeva od 100 do –95 s dekrementom 5.
#include <stdio.h>
main()
{
int i;
for(i=100; i>-100; i=i-5) printf("%d", i);
}
```

Oba primjera pokazuju da uslov mora biti na početku. Ako je laž, petlja se uopće ne vrti.

Primjer:

```
#include <stdio.h>
main()
{
int x, y;
x = 10;
for(y=10; y!=x; ++x) printf("%d", y);
/* this statement will not execute */
}
```

Neke varijacije petlje

Funkcija **kbhit()** – vraća odgovor laž ako se ne pritisne tipka, a istina ako je tipka pritisnuta.

Primjer:

```
#include <stdio.h>
#include <conio.h>
main()
{
int i;
/* print numbers until a key is pressed */
for(i=0; !kbhit(); i++) printf("%d", i);
}
```

.Tipka se pritisne, istina se vraća, !kbhit() je laž i petlja se zaustavlja.
.Tipka nije pritisnuta, !kbhit() je istina, petlja se nastavlja.

switch naredba

Naredba selekcije, odabira. Testira varijablu i niz integer ili character konstanti. Ako nijedan nije ispunjen izvršava se default.

Opći oblik: **switch** (variable) {
case constant 1 : statement sequence; break;
case constant 2 : statement sequence; break;
…
default : statement sequence }

Važno za switch:
- switch za razliku od if testira samo jednakost
- u istoj switch naredbi ne može biti dva case-a s istom vrijenošču
- switch je brži od niza if-ova
- dijelovi naredbi pridruženi svakom case-u nisu blokovi

Primjer:

Ako nema naredbe brek izvođenje se nastavlj na sljedeći case

```
include <stdio.h>
main()
{
int i;
for(i=0; i<5; i++){
switch(i){
case 0: printf(" 1\n");
case 1: printf(" 2\n");
case 2: printf(" 3\n");
case 3: printf(" 4\n");
case 4: printf(" 5\n");
}
printf("\n");
}
}
```

Mogu postojati i prazni case

```
switch ( i ) {
case 1 :
case 2 :
case 3 : radi();
break;
case 4 : ne_radi();
break;
}
```

Višestruki switch

```
switch (x) {
case 1 :
switch (y) {
case 0 : printf (“divide by zero”);
brek;
case 1 : process (x, y);
}
break;
case 2 : …
```

while petlja

Opći oblik:

while (uvjet) naredba;

Primjer:

```
#include <stdio.h>
main()
{
char ch;
ch = 1;
while(ch!=0){
printf("%c", ch);
ch++;
}
}
```

Na početku petlje testira se istinitost uslova, što znači da se petlja ne mora izršiti.

do / while petlja

Uslov se testira na kraju petlje, pa će se uvijek izvršiti barem jedanput.

Opći oblik:

```
do {
    naredbe;
    } while (uslov);
```

Zagrada se piše da se naredba ne pobrka s while naredbom.

Primjer:

```
#include <stdio.h>
main()
{
int num;
do{
scanf("%d", &num);
}while(num!=100);
}
```

continue

```
#include <stdio.h>
main()
{
int x;
for(x=0; x<=100; x++){
if(x%2)continue;
printf("%d", x);
} }
```

break

Break zaustavla najbližu petlju. Upotrebljen u naredbi switch samo djeluje na taj switch, a ne na petlju u kojoj se nalazi.

Primjer:

```
#include <stdio.h>
main()
{
int t, count;
for(t=0; t<100; t++){
count = 1;
for( ; ; ){
printf("%d", count);
count++;
if(count==10)break;
}
}
}
```

VJEŽBA

1. Napisati funkciju s imenom max koja vraća vrijednost većeg od dva cjelobrojna broja.

```
max (int a, int b)
    {
    if (a>b) return(a);
    else return(b);
    }
```

2. Napisati funkciju s imenom look_up() koja koristi cjelobrojni argument i vraća prema: ARGUMENT RETURN

1 a
2 b
3 c
4 d
inače vraća nulu.

```
look_ap(int i)
{
switch(i){
case 1 : return('a');
case 2 : return('b');
case 3 : return('c');
case 4 : return('d');
default : return ('0');
}
```

3. Zadatak 2 riješiti pomuću višestukog if

```
look_up(int i)
{
if(i==1) return('a');
else if(i==2) return('b');
else if(i==3) return('c'),
else if(i==4) return('d');
else return('0');
}
```

4. Napisati program koji uzima broj s tastature i ispisuje poruku "hello" onoliko puta koliko i broj.

```
#include <stdio.h>
main()
{
int t;
printf(" Enter a number: ");
scanf("%d", &t);
for( ;t>0;t--) printf(" HELLO\n");
}
```

ZADACI ZA VJEŽBANJE I OCJENJIVANJE C++

Tipovi podataka u C++ jeziku

- **unsigned short int** 2 byta od 0 do 65 535
- **short int** 2 bayta od –32 768 do 32 767
- **unsigned long int** 4 bayta od 0 do 4 294 967 295
- **long int** 4 bayta od –2 147 483 648 do 2 147 483 647
- **int (16 bit)** 2 bayta od –32 768 do 32 767
- **int (32 bit)** 4 bayta od –2 147 483 648 do 2 147 483 647
- **unsigned int (16 bit**) 2 bayta 0 do 65 535
- **unsigned int (32 bit)** 2 bayta od 0 do 4 294 967 295
- **char** 1 bayt 256 karakternih vrijednosti u ASCII standardu
- **float** 4 bayta 1.2e-38 do 3.4e38
- **double** 8 bayta 2.2e-308 do 1.8e308

Dodatak: Prikaz svih relacionih operatora u jeziku C++

Naziv	Operator
Jednako	==
Nije jednako	!=
Veće	>
Veće ili jednako	>=
Manje	<
Manje ili jednako	<=

LOGIČKE FUNKCIJE

I	**&&**
ILI	**|| - AltGr+w**
NE	**!**

Specijalni znakovi za znakovne nizove:

\t – horizontalni tab
\v – vertikalni tab
\n – prelazak u novi red
\a - signal
\b – backspace
\r – početak reda
\0 – završetak znakovnog niza
\? - upitnik
\f – nova stranica

1. **Napisati program koji sa tastature učitava dva cijela broja, a na izlazu daje zbir, razliku, proizvod , količnik ta dva broja, kao i kub prvog i korijen drugog broja . Uzeti samo cijele dijelove riješenja**

I način

```
#include<iostream.h>
#include<conio.h>
#include<math.h>  /* biblioteka podataka za matematicke funkcije*/
int main()
{
   int x,y,z,r,p,kub;
   float k,kor;
   cout<<"unesi prvi broj : ";
   cin>>x;
   cout<<"unesi drugi broj : ";
   cin>>y;
   z=x+y;
   r=x-y;
   p=x*y;
   kor=sqrt(y);    /* funkcija drugog korijena*/
   k=x/y;
   kub=pow(x,3);              /* pow(x,y)-funkcija stepena x^y */
   cout<<"zbri je : "<<z<<endl;
   cout<<"razlika je : "<<r<<endl;
   cout<<"proizvod je : "<<p<<endl;
   cout<<"kolicnik je : "<<k<<endl;
   cout<<"kub prvog broja je : "<<kub<<endl;
   cout<<"korijen drugog broja je : "<<kor<<endl;
   getch();
}
```

II način

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
   int x,y;
   cout<<"unesi prvi broj : ";
   cin>>x;
   cout<<"unesi drugi broj : ";
   cin>>y;
   cout<<"zbri je : "<<x+y<<endl;
   cout<<"razlika je : "<<x-y<<endl;
   cout<<"proizvod je : "<<x*y<<endl;
   cout<<"kolicnik je : "<<x/y<<endl;
   cout<<"kub prvog broja je : "<<pow(x,3)<<endl;
   cout<<"korijen drugog broja je : "<<sqrt(y)<<endl;
   getch(); }
```

2. Napisati program koji učitava tri broja i ispisuje njihov proizvod .

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int x,y,z;
     cout<<"unesi prvi broj : ";
     cin>>x;
     cout<<"unesi drugi broj : ";
     cin>>y;
     cout<<"unesi treci broj : ";
     cin>>z;
     cout<<"proizvod je : "<<x*y*z<<endl;
        getch();  }
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int x,y,z,p;
     cout<<"unesi prvi broj : ";
     cin>>x;
     cout<<"unesi drugi broj : ";
     cin>>y;
     cout<<"unesi treci broj : ";
     cin>>z;
     p=x*y*z;
     cout<<"proizvod je : "<<p<<endl;
        getch();  }
```

3. napisati program koji učitava broj sekundi (cijeli brojevi)i pretvara ih u dane, sate i minute.

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int s,dan,sat,min;
   cout<<"unesi sekunde : ";
   cin>>s;
   dan=s/86400;                        /* dan ima 86400 sekundi*/
   cout<<" dana ima : "<<dan<<endl;
   sat=(s-86400*dan)/3600;
   cout<<" sati ima : "<<sat<<endl;
   min=((s-86400*dan)-(3600*sat))/60;
   cout<<"minuta ima : "<<min<<endl;
   s=((s-86400*dan)-(3600*sat)-(60*min));
   cout<<"ostalo je sekundi: "<<s<<endl;
    getch(); }
```

4. **Napisati program koji učitava stranicu kvadrata i ispisuje njegovu površinu i obim**

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int a,p,o;
   cout<<"unesi stranicu a : ";
   cin>>a;
   p=a*a;
   o=4*a;
   cout<<"Povrsina je : "<<p<<endl;
   cout<<"Obim je : "<<o<<endl;
   getch();
}
```

5. **Napisati program koji učitava stranicu pravougaonika i ispisuje njegovu površinu i obim**

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int a,b,p,o;
   cout<<"unesi stranicu a : ";
   cin>>a;
   cout<<"unesi stranicu b : ";
   cin>>b;
   p=a*b;
   o=2*a+2*b;
   cout<<"Povrsina je : "<<p<<endl;
   cout<<"Obim je : "<<o<<endl;
   getch(); }
```

6. **Napisati program koji za učitanu vrijednost poluprečnika ispisuje površinu i obim kružnice**

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int r;
   float p,o;
   cout<<"unesi poluprecnik r : ";
   cin>>r;
   p=r*r*3.14;;
   o=2*r*3.14;
   cout<<"Povrsina je : "<<p<<endl;
   cout<<"Obim je : "<<o<<endl;
   getch(); }
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int r;
   float p,o;
   const float Pi=3.14;          /* deklaracija constante Pi koja je 3,14 */
   cout<<"unesi poluprecnik r : ";
   cin>>r;
   p=r*r*Pi;
   o=2*r*Pi;
   cout<<"Povrsina je : "<<p<<endl;
   cout<<"Obim je : "<<o<<endl;
   getch();
}
```

7. Napisati program koji učitava redni broj mjeseca i ispisuje koliko taj mjesec ima dana.

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int a;
   cout<<"unesi redni broj mjeseca : ";
   cin>>a;
  if (a==1)
        { cout<<" Januar ima 31 dan";}
        else
        if (a==2)
        { cout<<" Februar ima 29 ili 29 dana";}
        else
        if (a==3)
        { cout<<" Mart ima 31 dan";}
        else
        if (a==4)
        { cout<<" April ima 30 dana";}
        else
        if (a==5)
        { cout<<" Maj ima 31 dan";}
        else
        if (a==6)
        { cout<<" Juni ima 30 dana";}
        else
        if (a==7)
        { cout<<" Juli ima 31 dan";}
        else
        if (a==8)
        { cout<<" August ima 31 dan";}
```

```
        else
        if (a==9)
        { cout<<" Septembar ima 30 dana";}
        else
        if (a==10)
        { cout<<" Oktobar ima 31 dan";}
        else
        if (a==11)
        { cout<<" Novembar ima 30 dana";}
        else
        if (a==12)
        { cout<<" decembar ima 31 dan";}
        else
          { cout<<" unijeti broj nije redni broj mjeseca";}

    getch();
}
```

II način

```
/* Program radi koristenjem izraza za visestruko grananje
switch-case struktura . Ako se ulazni podatak npr a, deklarise kao char
onda u case izrazu se stavlja u apostrofe npr '1'  a ne (1)   */

#include<iostream.h>
#include<conio.h>
int main()
{
   int a;
    cout<<"unesi redni broj mjeseca : ";
    cin>>a;
   switch(a)
   {case (1):  {  cout<<" Januar ima 31 dan";break;getch();}
   case (2) :{ cout<<" Februar ima 29 ili 29 dana";break;getch();}
   case (3) :{ cout<<" Mart ima 31 dan";break;getch();}
   case (4) :{ cout<<" April ima 30 dana";break;getch();}
   case (5) :{ cout<<" Maj ima 31 dan";break;getch();}
   case (6) :{ cout<<" Juni ima 30 dana";break;getch();}
   case (7) :{ cout<<" Juli ima 31 dan";break;getch();}
   case (8) :{ cout<<" August ima 31 dan";break;getch();}
   case (9) :{ cout<<" Septembar ima 30 dana";break;getch();}
   case (10) :{ cout<<" Oktobar ima 31 dan";break;getch();}
   case (11) :{ cout<<" Novembar ima 30 dana";break;getch();}
   case (12) :{ cout<<" decembar ima 31 dan";break;getch();}
   default :  { cout<<" unijeti broj nije redni broj mjeseca";break;}
}
    getch();
}
```

8. **Napisati program koji učitava dva broja x i y, te ispisuje poruku da li je x>y ili x<y ili x=y**

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int x,y;
   cout<<"unesi x : ";
   cin>>x;
   cout<<"unesi y : ";
   cin>>y;
   if(x>y)
   {cout<<" x je veci od y";}
   else
   if(x<y)
   {cout<<" x je manji od y";}
   else
   {cout<<" x je jednak y";}
   getch();
}
```

9. **napisati program koji se za unešena dva cijela broja , na izlazu daje zbir ta dva broja ako je prvi broj manji od drugog, razlika ako je prvi broj veći od drugog ili proizvod ako su oba broja ista.**

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int x,y;
   cout<<"unesi x : ";
   cin>>x;
   cout<<"unesi y : ";
   cin>>y;
   if(x>y)
   {cout<<" razlika je "<< x-y;}
   else
   if(x<y)
   {cout<<" zbir je " <<x+y;}
   else
   {cout<<" proizvod je "<<x*y;}
   getch();
}
```

10. napisati program koji za unešeni cijeli broj ispituje i ispisuje da li je broj veći, manji ili jednak broju 10.

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int x;
    cout<<"unesi x : ";
    cin>>x;
     if(x>10)
    {cout<<" broj je veci od broja 10  ";}
    else
    if(x<10)
    {cout<<" broj je manji od broja 10 ";}
    else
    {cout<<" broj je jednak broju 10 ";}
    getch();
}
```

11. Napisati program koji učitava tri broja i ispisuje ih od manjeg ka većem

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int x,y,z;
    cout<<"unesi x : ";
    cin>>x;
    cout<<"unesi y : ";
    cin>>y;
    cout<<"unesi z : ";
    cin>>z;
    if((x>y)&&(y>z)&&(x>z))
    {cout<<" poredak je ";
        cout<<z;
        cout<<y;
        cout<<x;}
   else
   if((x>y)&&(y<z)&&(x>z))
    {cout<<" poredak je ";
        cout<<y;
        cout<<z;
        cout<<x;}
    else
    if((x<y)&&(y>z)&&(x>z))
        {cout<<" poredak je ";
```

```
        cout<<z;
        cout<<x;
        cout<<y;}
    else
    if((x>y)&&(y<z)&&(x<z))
        {cout<<" poredak je ";
        cout<<y;
        cout<<x;
        cout<<z;}
    else
    if((x<y)&&(y<z)&&(x<z))
        {cout<<" poredak je ";
        cout<<x;
        cout<<y;
        cout<<z;}
    else
       {cout<<" poredak je ";
        cout<<x;
        cout<<z;
        cout<<y;}
    getch();
}
```

12. Napisati program koji učitava tri broja i ispisuje ih od većeg ka manjem

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int x,y,z;
    cout<<"unesi x : ";
    cin>>x;
    cout<<"unesi y : ";
    cin>>y;
    cout<<"unesi z : ";
    cin>>z;
    if((x>y)&&(y>z)&&(x>z))
    {cout<<" poredak je ";
        cout<<x;
        cout<<y;
        cout<<z;}
   else
   if((x>y)&&(y<z)&&(x>z))
    {cout<<" poredak je ";
        cout<<x;
        cout<<z;
        cout<<y;}
    else
```

```
    if((x<y)&&(y>z)&&(x>z))
       {cout<<" poredak je ";
       cout<<y;
       cout<<x;
       cout<<z;}
    else
    if((x>y)&&(y<z)&&(x<z))
       {cout<<" poredak je ";
       cout<<z;
       cout<<x;
       cout<<y;}
    else
    if((x<y)&&(y<z)&&(x<z))
       {cout<<" poredak je ";
       cout<<z;
       cout<<y;
       cout<<x;}
    else
      {cout<<" poredak je ";
       cout<<y;
       cout<<z;
       cout<<x;}
    getch(); }
```

13. Napisati program koji sabira sve dvocifrene brojeve

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,s=0;
    for (i=10;i<=99;i++)
    s=s+i;
    cout<<" suma je :"<<s;
    getch();
}
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i=10,s=0;
    while (i<=99)
    {
    s=s+i;
    i++;}
    cout<<" suma je :"<<s;
    getch(); }
```

III način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i=10,s=0;
   do
   {s=s+i;
   i++;}
   while (i<=99);
   cout<<" suma je :"<<s;
   getch();
}
```

14. Napisati program koji ispisuje sve parne brojeve od 2 do 100

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i;
   for (i=2;i<=100;i=i+2)
     cout<<i<<endl;
   getch();  }
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i;
   for (i=1;i<=100;i++)
     if(i%2==0)
     {cout<<i<<endl;}

   getch(); }
```

III način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i=1;
   while(i<=100)
     {if(i%2==0)
     {cout<<i<<endl;}
     i++;
     }
   getch(); }
```

15. Napisati program koji ispisuje tablicu množenja brojem n za prvih deset prirodnih brojeva. Korisnik sam treba unijeti vrijednost broja n.

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i,n;
   cout<<" unesi prirodan broj n ";
    cin>>n;
   for(i=1;i<=10;i++)
     {cout<<i<<" * "<<n<<" = "<<i*n<<endl;}
   getch();
}
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i=1,n;
   cout<<" unesi prirodan broj n ";
    cin>>n;
   while(i<=10)
     {
     cout<<i<<" * "<<n<<" = "<<i*n<<endl;
     i++;}
   getch();
}
```

16. Napisati program koji sabira sve neparne brojeve od 1 do 9999

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i;
    long int s=0;
    for (i=1;i<=9999;i=i+2)
      s=s+i;
      cout<<" suma je "<<s;
    getch();
}
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i;
   long int s=0;
   for (i=1;i<=9999;i++)
     if(i%2==1)
     {s=s+i;}
     cout<<" suma je "<<s;
   getch();
}
```

III način

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i=1;
   long int s=0;
   while(i<=9999)
     {if(i%2==1)
     {s=s+i;}
     i++;}
     cout<<" suma je "<<s;
   getch();
}
```

17. Napisati program koji učitava prirodne brojeve sve dok se ne učita nula i ispisuje najveći učitani broj

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i,n,s=0;
   while (n!=0)
   {
   cout<<" unesi prirodan broj n ";
    cin>>n;
   if(s<=n)
   {s=n;}
}
     cout<<"najveci broj je "<<s<<endl;
   getch();
}
```

18. Napisati program kojim se daje suma prvih 100 brojeva

I način

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i=1,s=0;
    while (i<=100)
    {
    s=s+i;
    i++;
}
     cout<<"Suma je  "<<s<<endl;
    getch();
}
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i=1,s=0;
   for(i=1;i<=100;i++)
    s=s+i;
    cout<<"Suma je  "<<s<<endl;
    getch();
}
```

19. Napisati program kojim za unešenih 10 brojeva se daje suma pozitivnih i suma negativnih brojeva

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,sp=0,sn=0;
   for(i=1;i<=10;i++)
   {cout<<" Unesi broj: ";
   cin>>n;
   if (n>0)
   {sp=sp+n;}
   else sn=sn+n;}
     cout<<"Suma pozitivni je  "<<sp<<endl;
    cout<<"Suma negativni je  "<<sn<<endl;
    getch(); }
```

II način

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i=1,n,sp=0,sn=0;
   while(i<=10)
   {cout<<" Unesi broj: ";
   cin>>n;
   if (n>0)
   {sp=sp+n;}
   else sn=sn+n;
   i++;   }
     cout<<"Suma pozitivni je  "<<sp<<endl;
    cout<<"Suma negativni je  "<<sn<<endl;
    getch(); }
```

20. Napisati program kojim za unešenih 10 brojeva se na izlazu daje suma parnih i suma neparnih brojeva

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n,sp=0,sn=0;
  for(i=1;i<=10;i++)
  {cout<<" Unesi broj: ";
  cin>>n;
  if (n%2==0)
  {sp=sp+n;}
  else sn=sn+n;}
    cout<<"Suma parnih je  "<<sp<<endl;
   cout<<"Suma neparnih je  "<<sn<<endl;
   getch();  }
```

21. Napisati program kojim se za unešenih 10 brojeva na izlazu daje suma parnih negativnih i suma neparnih pozitivnih brojeva

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n,sp=0,sn=0;
  for(i=1;i<=10;i++)
  {cout<<" Unesi broj: ";
  cin>>n;
  if ((n%2==0)&&(n<0))
  {sp=sp+n;}
  else
  if((n%2==1)&&(n>0))
  {sn=sn+n;}
  }
    cout<<"Suma parnih negativnih je  "<<sp<<endl;
   cout<<"Suma neparnih pozitivnih je  "<<sn<<endl;    getch(); }
```

22. Napisati programkoji učitava x i daje ispis kvadratnih korijena brojeva x, x+1, x+2, x+3, x+4.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
   int x;
   cout<<"unesi x : ";
   cin>>x;
   cout<<endl;
   if (x>0)
   {cout<<" korijen od x je : "<<sqrt(x)<<endl;}
   else
   {cout<<" nema trazenog korijena od x"<<endl;}
    if (x+1>0)
   {cout<<" korijen od x+1 je : "<<sqrt(x+1)<<endl;}
   else
   {cout<<" nema trazenog korijena od x+1"<<endl;}
     if (x+2>0)
   {cout<<" korijen od x+2 je : "<<sqrt(x+2)<<endl;}
   else
   {cout<<" nema trazenog korijena od x+2"<<endl;}
     if (x+3>0)
   {cout<<" korijen od x+3 je : "<<sqrt(x+3)<<endl;}
   else
   {cout<<" nema trazenog korijena od x+3"<<endl;}
     if (x+4>0)
   {cout<<" korijen od x+4 je : "<<sqrt(x+4)<<endl;}
   else
   {cout<<" nema trazenog korijena od x+4"<<endl;}
   getch();
}
```

23. Napisati program kojim za unešenu vrijednost n na izlazu se ispisuje kvadrate brojeva od broja 1 do unešenog broja

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
   int n,i;
   cout<<"unesi n : ";
   cin>>n;
   cout<<endl;
   for(i=1;i<=n;i++)
   cout<<"kvadrat broja "<<i<<" je "<<pow(i,2)<<endl;    getch();  }
```

24. Napisati program kojim za unešeni broj , na izlazu se ispisuje da li je paran ili neparan

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int br;
    cout<<"unesi broj : ";
    cin>>br;
    cout<<endl;
    if(br%2==0)                    /* oznaka % uzima ostatak dijeljenja*/
    { cout<<" broj je paran"<<endl;}
    else cout<<"broj je neparan"<<endl;
    getch();
}
```

25. Napisati program za ispis proizvoda prvih 150 brojeva.

```
#include<iostream.h>
#include<conio.h>
int main()
{
  double i,p=1;

    for(i=1;i<=150;i++)
    p=p*i;
    cout<<"proizvod prvih 150 brojeva je "<<p;
    getch();
}
```

26. Napisati program za izračnavanje izraza s=1+1*2+1*2*3+…+1*2*…*n

```
#include<iostream.h>
#include<conio.h>
int main()
{
  double i,p=1,s=0,n;
  cout<<"unesi granicu izraza n";
  cin>>n;
    for(i=1;i<=n;i++)
    {p=p*i;
    s=s+p;
}
    cout<<"Ukupna suma izraza je "<<s;
    getch();
}
```

27. Napisati program za izračunavanje izraza P=5*(1+2)/1*(1+2+3)/2*…*(1+2+3+…+n)/(n-1)

```
#include<iostream.h>
#include<conio.h>
int main()
{
  double i,p=5,s=1,n;
  cout<<"unesi granicu izraza n";
  cin>>n;
    for(i=2;i<=n;i++)
    {s=s+i;
    p=p*(s/(i-1));
 }
    cout<<"Ukupni proizvod izraza je "<<p;
    getch();
}
```

28. Napisati program koji ispisuje kvadrate prirodnih brojeva od 50 do 20

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
    int i;
    for(i=50;i>=20;i--)
    cout<<"kvadrat broja "<<i<<" je "<<pow(i,2)<<endl<<endl;
    getch();
}
```

29. Napisati program za ispis zbira pozitivnih prirodnih brojeva manjih od 500 a djeljivi sa 3, i navedi koliko tih brojeva ima.

```
#include<iostream.h>
#include<conio.h>

int main()
{
    int i,s=0,k=0;
    for(i=500;i>0;i--)
    {if(i%3==0)
    {s=s+i;
    k=k+1;}}
    cout<<"Zbir je"<<s<<endl;
    cout<<" ukupno u zbiru ucestvuje "<<k<<" brojeva ";
    getch();
}
```

30. Napisati program za određivanje razlike između zbira prvih 10 parnih brojeva i zbira prvih deset neparnih brojeva

```
#include<iostream.h>
#include<conio.h>

int main()
{
   int i,sp=0,sn=0,r=0;
   for(i=1;i<=10;i++)
   {if(i%2==0)
   {sp=sp+i;}
   else
   {  sn=sn+i;}
   }
   r=sp-sn;
   cout<<"razlika je "<<r<<endl;
      getch();
}
```

31. Napisati program za ispis zbira dva broja ako je jedan od njih manji od nule, a ako su oba veća od nule onda ispiši njihove razlike

```
#include<iostream.h>
#include<conio.h>

int main()
{
   int x,y,z,r1,r2;
   cout<<"unesi prvi broj : ";
   cin>>x;
   cout<<"unesi drugi broj : ";
   cin>>y;
   if((x<0)||(y<0))
   {z=x+y;
   cout<<"zbir je "<<z<<endl;
   }
   else
   if((x>0)&&(y>0))
   {
   r1=x-y;
   r2=y-x;
   cout<<"razlika "<<x<<" - "<<y<<" je "<<r1<<endl;
   cout<<"razlika "<<y<<" - "<<x<<" je "<<r2<<endl;
   }
     getch();
}
```

32. Napiši program za ispis prve i treće cifre trocifrenog broja

```
#include<iostream.h>
#include<conio.h>
int main()
{
   int i;
   char znak[3];                  /* broj se unosi kao niz od 3 znaka */
   cout<<"unesi trocifreni broj ";
   for (i=1;i<=3;i++)
   cin>>znak[i];
   cout<<"prva i treca cifra je : ";
    for (i=1;i<=3;i=i+2)
   cout<<znak[i];
   getch();
}
```

33. Napisati program za ispis neke riječi u obrnutom redoslijedu koristiti max 50 znakova (npr. MIR kao RIM)

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
int main()
{
   int i,j=0;
   char niz[50];
   cout<<"unesi znakovni niz ";
   gets(niz);                              /* unos elemenata znakovnog niza */
   cout<<endl<<"ispis u normalnom obliku"<<endl;
   for (i=0;niz[i]!='\0';i++)    /* unos se prekida pritiskom na enter tipku */
   {cout<<niz[i];
   j++;
}
   cout<<endl<<endl<<"ispis u obrnutom redoslijedu"<<endl;
   for (i=j;i>=0;i--)
   cout<<niz[i];
   getch();
}
```

34. Ako je januar prvi mjesec u godini, a decembar dvanaesti, potrebno je za zadani broj od 1 do 12 ispisati ime mjeseca u godini

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int a;
    cout<<"unesi redni broj mjeseca : ";
    cin>>a;
   if (a==1)
          { cout<<" Januar";}
          else
          if (a==2)
          { cout<<" Februar a";}
          else
          if (a==3)
          { cout<<" Mart ";}
          else
          if (a==4)
          { cout<<" April a";}
          else
          if (a==5)
          { cout<<" Maj ";}
          else
          if (a==6)
          { cout<<" Juni ";}
          else
          if (a==7)
          { cout<<" Juli n";}
          else
          if (a==8)
          { cout<<" August";}
          else
          if (a==9)
          { cout<<" Septembar";}
          else
          if (a==10)
          { cout<<" Oktobar ";}
          else
          if (a==11)
          { cout<<" Novembar ";}
          else
          if (a==12)
          { cout<<" decembar ";}
          else
            { cout<<" unijeti broj nije redni broj mjeseca";}
    getch();
}
```

ZA VJEŽBU ZADATAK URADITI CASE STRUKTUROM

35. Napisati program za ispis zbira cifara unesenog broja

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int i,s;
char niz[50];
cout<<"unesi broj ";
gets(niz);
for (i=0;niz[i]!='\0';i++)
s=s+niz[i]-48;
 /*Broj se unosi kao niz karaktera te se sa -48 znakovi pretvaraju u broj*/
cout<<" suma cifara je "<<s;
getch();
}
```

36. Napisati program koji za zadane dužine stranica trougla ispituje da li je trougao pravougli

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
   int a,b,c;

   cout<<"unesi stranicu a : ";
   cin>>a;
   cout<<"unesi stranicu b : ";
   cin>>b;
    cout<<"unesi stranicu c : ";
   cin>>c;
   if (c==sqrt(pow(a,2)+pow(b,2)))
     cout<<"trougao je pravougli"<<endl;
     else
   cout<<"trougao nije pravougli "<<endl;
   getch();
}
```

37. Napisati program za ispis prvih n neparnih brojeva, korisnik unosi vrijednost n.

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n;
   cout<<" Unesi granicu n :";
    cin>>n;
    cout<<" Neparni brojevi su :"<<endl;
   for(i=1;i<=n;i++)
   if (i%2==1)
   cout<<i<<endl;
      getch();  }
```

38. Unesi broj n (ne veći od 100)a zatim n pozitivnih članova niza. . Ispisati one članove niza koji su dvocifreni brojevi

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n,niz[100];
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=1;i<=n;i++)
    cin>>niz[i];
    cout<<" dvocifreni pozitivni brojevi u nizu su : "<<endl;
    for(i=1;i<=n;i++)
   if ((niz[i]>9)&&(niz[i]<100))
   cout<<niz[i]<<endl;
   getch();
}
```

39. Unesi broj n a zatim n članova niza. . Ispisati one članove niza koji su djeljivi sa brojem 5

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n,niz[100];
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=1;i<=n;i++)
    cin>>niz[i];
    cout<<" brojevi djeljivi sa 5 u nizu su : "<<endl;
    for(i=1;i<=n;i++)
   if (niz[i]%5==0)
   cout<<niz[i]<<endl;   getch();  }
```

40. Unesi broj n a zatim n članova niza. Sabrati sve članove niza koji se nalaze na parnim mjestima u nizu

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,niz[100],s=0;
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=1;i<=n;i++)
    cin>>niz[i];
    cout<<endl;
    for(i=1;i<=n;i++)
   if (i%2==0)
   s=s+niz[i];
   cout<<" suma elemenata u nizu na parnim mjestima je " <<s<<endl;
   getch();
}
```

41. Unesi broj n a zatim n članova niza. . Izračunati zbir svih pozitivnih članova niza

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,niz[100],s=0;
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=1;i<=n;i++)
    cin>>niz[i];
    cout<<endl;
    for(i=1;i<=n;i++)
   if (niz[i]>0)
   s=s+niz[i];
   cout<<" suma pozitivnih elemenata u nizu je " <<s<<endl;
   getch();
}
```

42. Unesi broj n a zatim n članova niza. . Razvrstati sve članove niza od najmanjeg ka najvećem.

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,j,n,niz[100],pom;
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=0;i<n;i++)
      cin>>niz[i];
    cout<<endl;
      for(i=0;i<n-1;i++)
       for(j=0;j<n-1-i;j++)
   if (niz[j+1]<niz[j])
   {
       pom=niz[j];
       niz[j]=niz[j+1];
       niz[j+1]=pom;
   }
cout<<" elementi niza u rastucem poretku : "<<endl;
 for(i=0;i<n;i++)
 cout<<niz[i]<<"\t";   /* \t- pomak izpisa za vrijednost tabulatora*/
   getch();
}
```

43. Unesi broj n a zatim n članova niza. Ispisati najveći član niza

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,niz[100],naj;
   cout<<" Unesi granicu niza n :";
    cin>>n;
    cout<<" unesi elemente niza : "<<endl;
    for(i=1;i<=n;i++)
    cin>>niz[i];
    cout<<endl;
    naj=0;
    for(i=1;i<=n;i++)
   if (naj<niz[i])
   naj=niz[i];
   cout<<" najveci uneseni element niza je " <<naj<<endl;
   getch();
}
```

44. Napisati program koji za uneseno n ispisuje slijedeću sliku
X za n=4
XX
XXX
XXXX

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,j;
   cout<<" Unesi broj n :";
    cin>>n;

    for(i=1;i<=n;i++)
    {for(j=1;j<=i;j++)
    cout<<"X";
   cout<<endl;
}
     getch();
}
```

45. Napisati program koji za uneseno n ispisuje slijedeću sliku
X
XX n=4
XXX
XXXX

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int i,n,j,k;
   cout<<" Unesi broj n :";
    cin>>n;

    for(i=1;i<=n;i++)
    {for(j=0;j<=n-i;j++)
    cout<<" ";
     for(k=j;k<=n;k++)
     cout<<"X";
   cout<<endl;
}
     getch();
}
```

46. Napisati program koji za uneseno n ispisuje slijedeću sliku

```
    X
   XXX      n=3
  XXXXX
```

```
#include<iostream.h>
#include<conio.h>
int main()
{    int i,n,j,k;
   cout<<" Unesi broj n :";
    cin>>n;
    for(i=1;i<=n;i++)
    {for(j=1;j<=n-i;j++)
    cout<<" ";
     for(k=1;k<=(2*i-1);k++)
     cout<<"X";
   cout<<endl;
}
     getch();  }
```

47. kreiranje, upisivanje I čitanje iz datoteke

```
// Program DATOTEKE ver. 1.0
// program od korisnika trazi da unese ime datoteke, ista se formira na lokaciji
// izvrsnog fajla, i unutar nje se upisuju dva reda podataka koje se traze
// od korisnika
#include <fstream.h>
int main()
{
char ImeFajla[80];
char Sadrzaj[255]; // nizovi za ime datoteke i njen sadrzaj
char Sadrzaj1[255];
cout << "Unesite naziv fajla plus ekstenzija TXT: ";
cin >> ImeFajla;
ofstream fout(ImeFajla); // kreiranje i otvaranje datoteke
cout << "Unesite bilo sta: ";
cin.ignore(1,'\n'); // ignorisanje linije
cin.getline(Sadrzaj,255); // ucitavanje unosa sa tastature
fout << Sadrzaj << "\n"; // i upisivanje u datoteku
cout << "Unesite jos jednom bilo sta: ";
cin.getline(Sadrzaj1,255);
fout << Sadrzaj1 << "\n";
fout.close(); // zatvaranje datoteke
ifstream fin(ImeFajla); // otvaranje datoteke prije citanja
cout << "Ovo je sadrzaj fajla koji ste unijeli\n";
char ch;
while (fin.get(ch))
cout << ch;
cout << "\n***Kraj datoteke.***\n";
fin.close();   return 0;   }
```

DODATAK

Odnos između Pascala i C-a

Osnovni koncepti Pascala prisutni su i u jeziku C, s tim da C dopušta veća odstupanja. C je jezik koji s jedne strane omogućuje programeru jednostavan pristup do elementarnih strojnih elemenata računara, s jedne strane, a s druge povećanu efikasnost programera s druge, zahvaljujući odmaku od krute strukture programa u Pascalu
Neke od osnovnih razlika između Pascala i C-a možemo vidjeti u slijedećoj tablici :

	Pascal	**C**
sekvenca	BEGIN-END	{ }
selekcija	IF, CASE	if, switch
iteracija	FOR, WHILE, REPEAT-UNTIL	for, while, do-while
potprogrami	procedure i funkcije	funkcije
slogovi	RECORD	struct
razlikovanje malih/velikih slova	NE	DA, ključne riječi obavezno malim slovima
aritmetički operatori	+ - * / div mod	+ - * / % unarni operatori
pokazivači	^	*
reference	funkcija ADDR	&
operator pridruživanja	:=	= operatori obnavljajućeg pridruživanja
relacijski operatori	= < > <> <= >=	== < > != <= >=
logički operatori	and or not	&& || !
bitovni operatori	and or not shl shr	& | ~ << >>
operator dodjele tipa	ne postoji (koriste se funkcije)	postoji
uvjetni operator	ne postoji	?
tip logičkih izraza	boolean (logički)	int (cjelobrojni)
komentar	{ i } ili (* i *)	/* i */

Odnos između C i C++

Kao što je Pascal najtipičniji strukturno-orijentirani programski jezik, tako je C++ (uz Javu, koja, međutim, nasljeđuje većinu koncepata i sintaksu od C++) najtipičniji objektno-orijentirani programski jezik. C++ predstavlja proširenje programskog jezika C, u kojeg uvodi koncept objektno-orijentiranog programiranja. Jezik C ostao je i dalje uključen u jezik C++, kao njegov podskup. Kao takav, C++ podržava oba programska koncepta : koncept strukturnog programiranja i koncept objektno-orijentiranog programiranja.

Usprkos tome što programi pisani u C-u rade i u C++, C++ u dosta slučaja nudi drukčiji put rješavanja istih programskih zadataka.

Terminalski ulaz/izlaz

Jedna od najočitijih razlika između C i C++ je zamjena standardne biblioteke *stdio* bibliotekom *iostrea*m. Biblioteka *iostream* zamjenjuje sve mogućnosti biblioteke *stdi*o, odnosno, tri programska retka iz slijedećeg primjera daju identičan rezultat :

```
printf ("Danas je lijep dan.\n"); // C
cout << "Danas je lijep dan.\n"; // kombinirano rješenje
cout << "Danas je lijep dan." << endl; // C++
```

Za formatiranje izlaza mogu se koristiti manipulatori, koji su definirani unutar biblioteke *iomani*p.

Primjer :

```
float pi=3.14159;
cout << setprecision(4) << pi << endl; // ispisuje se 3.141
```

Ulazom se upravlja na sličan način, s tim da se koristi ulazni tok *cin* i operator izlučivanja iz toka (>>).

Primjer :

```
scanf ("%f",&pi); // C
cin >> pi; // C++
```

Datotetični ulaz/izlaz

Umjesto biblioteke *stdio* koristi se biblioteka *fstream* (također se mogu koristiti *ifstream* i *ofstrea*m). Slijedeći primjer u C-u upisuje dva reda teksta u izlaznu tekstualnu datoteku :

```
 FILE *dat;
```

```
dat = fopen ("tekst1.txt","wt");
fprintf (dat,"%s","Prvi red teksta!\n");
fprintf (dat,"%s","Drugi red teksta!\n");
fclose(dat);
```

Odgovaraju æ i primjer u C++ :

```
fstream dat;
dat.open ("tekst.txt",ios::out);
dat << "Prvi red teksta!" << endl;
dat << "Drugi red teksta!" << endl;
dat.close();
```

U drugom primjeru definiran je datotečni objekt *da*t, iz klase *fstream.* Za otvaranje datoteke korišten je funkcijski član *ope*n. Datoteka je otvorena u modu *out* (izlazna datoteka). Tekst se u datoteku upisuje korištenjem operatora umetanja u tok (<<).

Konstante

U C-u se za definiranje konstanti koristi pretprocesorska naredba #*define*.

Na primjer: **#define UKUPNO 50**
U tu svrhu u C++ se može koristiti i ključna riječ *const* : **const UKUPNO = 50;**
Također, C++ dopušta da pojedini argumenti funkcija budu konstantni, time se sprečava promjena njihove vrijednosti unutar funkcije.

Primjer :
```
void funkcija (const int a)
        {
        a=10; // Greška; ne može se promijeniti konstanta
        }
```

Preopterećenje funkcije

C++ omogućuje da više funkcija ima isti naziv, pod uslovom da su im liste parametara različite. Prema listi parametara kod poziva funkcije određuje se koja će funkcija biti pozvana.

Primjer :

```
#include <iostream.h>
void funkcija(){
cout << "Funkcija bez parametara!" << endl;
}
void funkcija(int a){
cout << "Cjelobrojni parametar a = " << a << endl;
}
void funkcija(float b, float c){
cout << "Realni parametar b = " << b << endl;
cout << "Realni parametar c = " << c << endl;
}
void main(){
funkcija();
funkcija(10);
funkcija(2.71,3.14); }
```

Ispisuje se :

Funkcija bez parametara!
Cjelobrojni parametar a = 10
Realni parametar b = 2.71
Realni parametar c = 3.14

Podrazumijevani argumenti funkcija

Prilikom poziva funkcije potrebno je navesti listu argumenata koja odgovara listi argumenata u zaglavlju funkcije. C++ dopušta da se neki od argumenata u pozivu funkcije izostave, tako da se umjesto njih koriste podrazumijevane vrijednosti.

Primjer:

```
#include <iostream.h>
void funkcija(int a, int b=5){
cout << "a = " << a << endl;
cout << "b = " << b << endl << endl;
}
void main(){
funkcija (3,4);
funkcija (10);
}
```

Ispisuje se :

a = 3
b = 4
a = 10
b = 5

U drugom pozivu funkcije izostavljen je drugi argument, umjesto kojeg se koristi podrazumijevani (b=5).

Alokacija memorije

C++ zamjenjuje C-ovu funkciju za alokaciju memorije *malloc* i funkciju za dealokaciju memorije *free* s *new* i *delete*. New i delete omogućuju alokaciju korisnički definiranih tipova jednako kao i preddefiniranih.

Primjer :

C:

```
int *pok;
pok = (int*)malloc(sizeof(int));
*pok = 10;
free (pok);
```

C++ :

```
int *pok;
pok = new int;
*pok = 10;
delete pok;
```